Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра обчислювальної техніки
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмний засіб для обробки та передачі аналітичних даних на платформі Android»

Виконав: студент 2-го курсу, групи 1КІ-20м
спеціальності: 123 - Комп'ютерна інженерія
(шифр і назва напряму підготовки)
_____Чурчун В. В.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. ОТ
_____Войцеховська О. В.
(прізвище та ініціали)
«___» _____ 2021 р.

Опонент: д.т.н., професор каф. МБІС
_____Яремчук Ю. Є.
(прізвище та ініціали)
«___» _____ 2021 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
(прізвище та ініціали)
«____» _____ 2021 р.

Вінниця ВНТУ 2021

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Рівень вищої освіти ІІ-й (магістерський)
Спеціальність – 123 – Комп'ютерна інженерія
Освітньо-професійна програма – 123 – Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
_____ 202**1** року

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Чурчуну Владиславу Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Програмний засіб для обробки та передачі аналітичних даних на платформі Android», керівник роботи Войцеховська Олена Валеріївна, к.т.н., доцент затверджені наказом вищого навчального закладу від "24" 09 2021 року №277

2. Строк подання студентом роботи ___ _____ 202_ року

3. Вихідні дані до роботи: проаналізувати сучасний стан в сфері аналітики даних та провести огляд існуючих бібліотек для збору аналітичних даних, розробити механізм збору та обробки аналітичних даних у додатках на операційній системі Android

4. Зміст текстової частини: вступ, аналіз сучасного стану в сфері аналітики даних та огляд існуючих бібліотек для збору аналітичних даних, методи та технології для збору й обробки аналітичних даних, програмна реалізація бібліотеки, економічна частина.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): частота відкриття додатків з персоналізованими та не персоналізованими даними, відсоткова залежність користувачів що відмовляються від додатків після першого використання, візуалізація роботи

вдосконаленого методу збору й обробки аналітичних даних, життєвий цикл фрагмента у системі Android, UML діаграма додатку.

6. Консультанти розділів роботи

Консультанти роботи наведені в таблиці 6.1.

Таблиця 6.1 – Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-4	Войцеховська О. В., к.т.н., доцент кафедри ОТ		
5	Кавецький В. В., к.т.н., доцент кафедри ЕПВМ		

7. Дата видачі завдання _____2021 року

8. Календарний план

Календарний план наведено в таблиці 8.1.

Таблиця 8.1 – Календарний план

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Формулювання мети та цілей для її досягнення	07.09.21 - 10.09.21	
2	Аналіз сучасного стану в сфері аналітики даних та огляд існуючих бібліотек для збору аналітичних даних	11.09.21 - 30.09.21	
3	Огляд методів та технологій для збору й обробки аналітичних даних	01.10.21 - 30.10.21	
4	Написання додатку	01.11.21 - 15.11.21	
5	Тестування створеного додатку	16.11.21 - 19.11.21	
6	Економічний аналіз роботи	20.11.21 - 28.11.21	
7	Оформлення додатків	29.11.21 - 30.11.21	
8	Оформлення пояснювальної записки	01.12.21 - 05.12.21	

Студент Чурчун В. В. _____ (підпис)

Керівник роботи Войцеховська О. В. _____ (підпис)

Консультант з економічної частини Кавецький В. В _____ (підпис)

АНОТАЦІЯ

УДК 004.4

Чурчун В. В. Програмний засіб для обробки та передачі аналітичних даних на платформі Android. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна інженерія, освітня програма — Комп'ютерна інженерія. Вінниця: ВНТУ, 2021. 117 с.

В даній магістерській кваліфікаційній роботі було проведено аналіз сучасного стану в сфері обробки аналітичних даних у мобільних додатках та створено програмний засіб для обробки та передачі аналітичних даних на системі Android. Даний додаток використовується в коді клієнтського додатку та дозволяє проводити збір аналітичних даних з мінімальною кількістю необхідного коду.

В аналітичному розділі було розглянуті актуальність розробки в сфері аналітики мобільних додатків, актуальні альтернативні рішення та обмеження, що накладає регламент GDPR на обробку особистих даних. У розділі теоретичних досліджень вдосконалено метод збору та обробки аналітичних даних з подальшим оглядом технологій, необхідних для його реалізації. В конструктивному розділі було створено додаток, що використовує розроблений метод та відповідає вимогам, вказаним у технічному завданні. У верифікаційному розділі проведено тестування з використанням тестового додатку.

Створений додаток відповідає стандартам регламенту GDPR та не порушує права користувача на обробку персональних даних. Проведені економічні розрахунки вказують на доцільність розробки.

Ключові слова: Android, Firebase Analytics, аналітика, аспектно-орієнтоване програмування, відслідковування.

ABSTRACT

Churchun V.V. Software for processing and transmission of analytical data on the Android platform. Master's thesis in specialty 123 — Computer Engineering, educational program — Computer Engineering. Vinnytsa: VNTU, 2021. - 117 p.

In this master's qualification work the current state in the field of analytical data processing in mobile applications was analyzed and a software tool for processing and transmitting analytical data on Android was created. This application is used in the code of the client application and allows the collection of analytical data with a minimum amount of required code.

The analytical section considered the relevance of development in the field of mobile application analytics, current alternative solutions and limitations imposed by the GDPR regulations on the processing of personal data. In the section of theoretical researches the method of collecting and processing of analytical data with the subsequent review of the technologies necessary for its realization is improved. In the design section, an application was created that uses the developed method and meets the requirements specified in the technical task. In the verification section, testing was performed using a test application.

The created application complies with the standards of the GDPR regulations and does not violate the user's right to personal data processing. Conducted economic calculations indicate the feasibility of development.

Keywords: Android, Firebase Analytics, analytics, aspect-oriented programming, tracking.

ЗМІСТ

ВСТУП

По мірі того, як світ вдосконалюється, дані стають ключем до конкурентних переваг. Таким чином здатність компанії конкурувати все більше буде залежати від того, наскільки добре вона може використовувати дані, застосовувати аналітику та впроваджувати нові технології. За даними Міжнародного інституту аналітики, до 2022 року підприємства, які використовують аналітичні дані, отримають 430 млрд доларів переваг у продуктивності порівняно з конкурентами, які відмовились від цього.

Отже, зрозуміло, що дані зараз є ключовим бізнес-активом, і вони революційно змінюють спосіб роботи компаній у більшості галузей. По суті, кожен бізнес, незалежно від розміру, тепер повинен бути бізнесом даних. І, якщо кожен бізнес є бізнесом даних, то кожен бізнес потребує надійної стратегії передачі даних.

Не менш актуально постає дана ситуація і у світі програмного забезпечення. За допомогою інформації про поведінку користувача в додатку, можна розвивати відповідні галузі програми, пріорітезувати тестування, а також відповідним чином керувати рекламною інтеграцією.

Операційна система Android є лідером на ринку мобільних операційних систем, що робить дослідження в даній галузі цілком доцільним. Більшість компаній використовує для збору аналітичних даних безкоштовні рішення, що надаються іншими підприємствами. Існує і кілька платних рішень, що надають більше функціональних можливостей та полегшують налаштування. Незважаючи на це, існуючі рішення не є ідеальними та мають ряд недоліків, найголовнішим з яких є відсутність інтеграції програмної бібліотеки з компонентами операційної системи та відсутність архітектурних рішень щодо автоматизації формування та збереження аналітичних даних.

Отже, розробка програмного засобу для обробки та передачі аналітичних даних на платформі Android є актуальною задачею.

Програмний засіб, що розробляється, матиме вигляд бібліотеки, сумісної з додатками на операційній системі Android, що дозволятиме формувати аналітичні події за допомогою мінімальної кількості коду та дозволятиме автоматизувати створення подій при використанні стандартних компонентів операційної системи Android.

Метою дослідження магістерської роботи є вдосконалення методу збору та передачі аналітичних даних, а також розробка відповідного додатку на операційній системі Android.

Задачі дослідження магістерської роботи:

— аналіз існуючих бібліотек для збору аналітичних даних, методів обробки та відправки даних;

— розробка удосконаленого методу для збору та передачі даних, використовуючи інтеграцію з ключовими компонентами системи Android;

— створення бібліотеки з реалізацією запропонованих ідей, що буде відповідати необхідним вимогам.

Об’єкт дослідження магістерської роботи — процес збору та передачі аналітичних даних в додатках на платформі Android.

Предмет дослідження магістерської роботи — методи покращення збору та передачі аналітичних даних зі сторони клієнтського додатку.

Методи дослідження магістерської роботи: методи інтеграції програмної бібліотеки з компонентами операційної системи Android, методи автоматизації формування аналітичних даних. У роботі використано принципи об’єктно-орієнтованого та аспектно-орієнтованого програмування для реалізації запропонованого підходу.

Наукова новизна отриманих результатів магістерської роботи полягає в розробці удосконаленого методу збору та відправки аналітичних даних у програмних додатках на платформі Android, який дозволяє зменшити затрати часу та роботи на реалізацію даного функціонального рішення в додатках.

Практичне значення одержаних результатів магістерської роботи: розроблено програмну бібліотеку, що інтегрується з основними компонентами операційної системи Android та за допомогою автоматизації полегшує роботу з аналітичними даними зі сторони клієнтського коду.

Апробація результатів кваліфікаційної роботи. За результатами роботи зроблено доповідь на Всеукраїнській науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2022)» [1].

1 АНАЛІЗ СУЧАСНОГО СТАНУ В СФЕРІ АНАЛІТИКИ ДАНИХ ТА ОГЛЯД ІСНУЮЧИХ БІБЛІОТЕК ДЛЯ ЗБОРУ АНАЛІТИЧНИХ ДАНИХ

1.1 Огляд ролі аналітичних даних у мобільних додатках

Мобільні додатки у відповідних магазинах постійно борються за видимість та увагу користувачів. Оскільки рівень конкуренції дуже високий, важливо мати стратегію розвитку додатку та інвестування необхідних коштів. Слід враховувати, що маркетинг потребує великої кількості коштів, тому потрібно максимально ефективно використовувати витрачені години та гроші.

Переглядаючи аналітичні дані, можна скласти схему, які канали для інвестування найкраще підходять власнику та його додатку, щоб можна було точно витратити більшу частину свого маркетингового бюджету.

Однак аналітика додатків — це не лише оптимізація магазинів додатків (App Store Optimization, ASO), а й опирання на пошукові системи магазину. Надійна стратегія ASO для високого рейтингу в результатах пошуку не менш важлива ніж успішна стратегія аналітики.

Спостерігаючи за тим, які канали забезпечують найвищу цінність за весь час користування (Lifetime Value, LTV), можна визначити, які з них приносять більший дохід або, можливо, збереження існуючих користувачів.

Важливою частиною розробки стратегії аналітики є можливість користувачів зв’язуватися з командою в потрібний час у всьому додатку. Це важливо, оскільки дозволяє покращити маркетингову стратегію в режимі реального часу. Очевидно, що без інтеграції аналітики у маркетингові канали не можна отримати результати з аналітики користувача додатку.

Завдяки аналітиці додатків тепер можливе створення індивідуальної цільової взаємодії з користувачами. Цільові push-повідомлення, доставлені конкретним людям, можуть збільшити середню частоту відкриття додатків в чотири рази, з 1,5% до 5,9%, у порівнянні з повідомленнями загального типу (рисунок 1.1) [2].

Аналітика додатків — це хороший спосіб познайомитись із новими користувачами, у тому числі з тими, хто випав із маркетингової воронки та може бути переведений на вищу стадію життєвого циклу. На основі зібраних даних можна встановити, хто завантажує програму та відкриває її знову після першого перегляду. Аналізуючи поведінку користувачів додатку та дані профілю, вносяться зміни до маркетингової стратегії, щоб завоювати цільову аудиторію, використовуючи персоналізовані повідомлення та інформацію про досвід роботи в додатку.

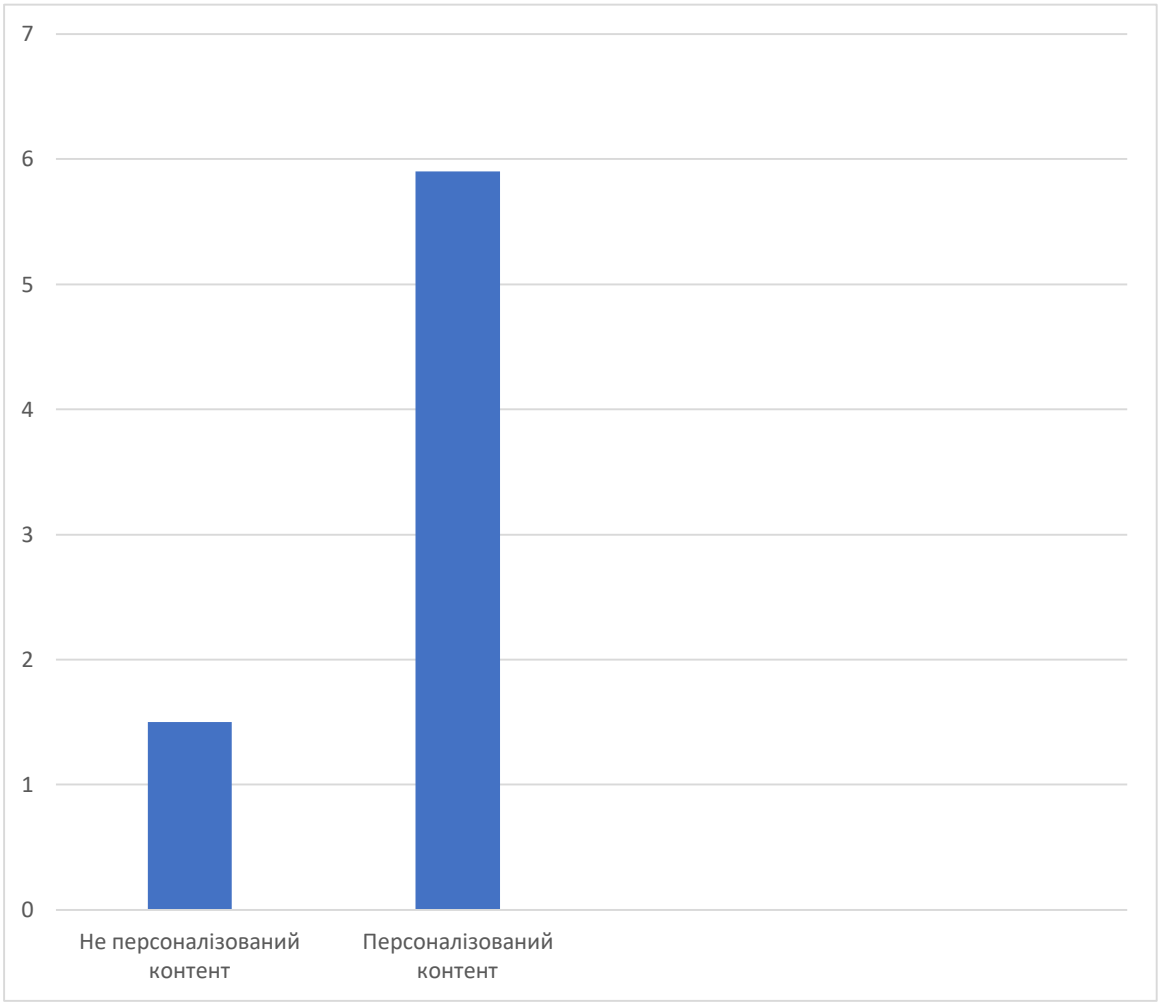

Рисунок 1.1 – Частота відкриття додатків з персоналізованими та не персоналізованими даними

Аналітика додатків дозволяє зрозуміти, що хочуть і потребують користувачі. Багато програм втрачають велику кількість користувачів через те, що розробники не прислуховуються до них. Робота не завершується, як тільки

користувач залучений. Після цього необхідно заглибитися в дані програми користувача та дізнатися, як змусити його залишитись і використовувати додаток. Забезпечуючи утримання користувачів, потрібно постійно оновлювати та вдосконалювати додаток на основі досвіду та думки користувачів.

Майже кожна четверта людина відмовляється від мобільних додатків лише після одного використання (рисунок 1.2), при цьому користувачі швидко виходять і більше ніколи не повертаються, щоб відкрити додаток. Тому це розуміння та зворотній зв'язок з користувачами є необхідним для збільшення залучення.

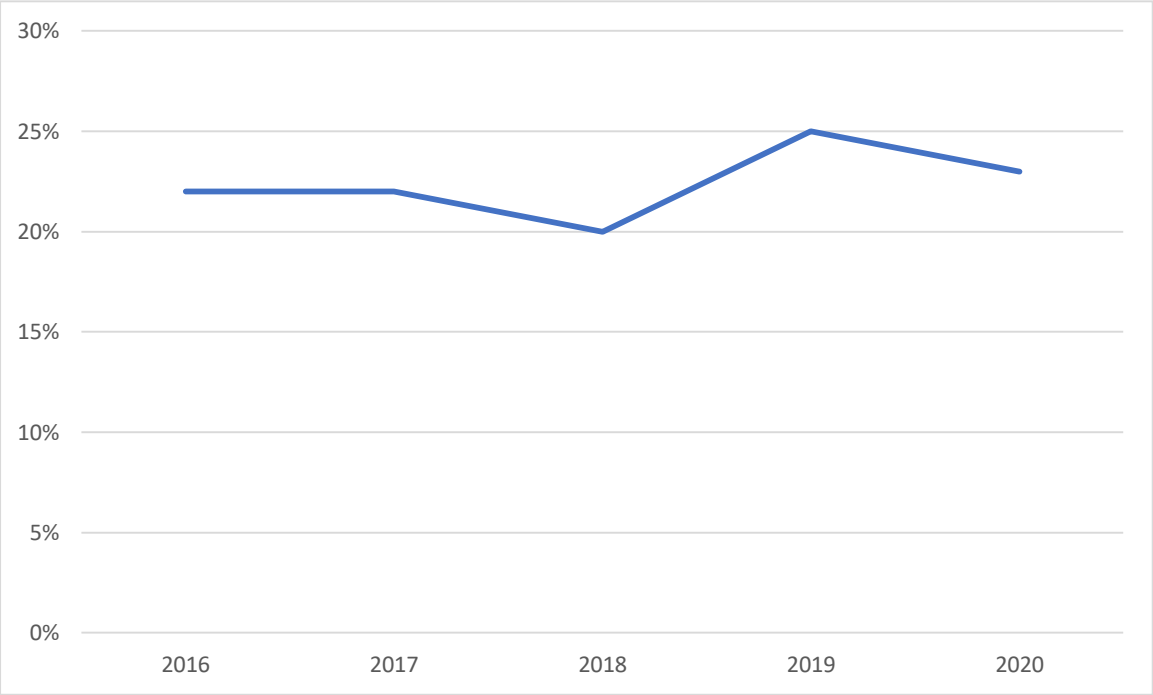

Рисунок 1.2 – Відсоткова залежність користувачів, що відмовляються від додатків після першого використання

У маркетологів існує велика кількість різних каналів, за допомогою яких вони можуть залучати нових користувачів програми. Однак з цим величезним вибором може виникнути плутанина щодо того, що найкраще для їх цільових користувачів. Аналітика додатків дозволяє побачити, чи витрачає власник свої гроші та час на правильні канали придбання чи ні. Визначення правильних

каналів є важливим для того, щоб витратити час та маркетинговий бюджет на свою користь.

Тож за допомогою аналітики додатків можна глибоко зануритися у поведінку користувачів і дізнатися, що їм подобається, а що ні. Все це можна зробити, не запитуючи їх, і тим самим позбавившись від життєво важливих годин, грошей та робочої сили, що дозволить витрачати час на інші важливі завдання.

1.2 Огляд системи Android

Android є найпопулярнішою операційною системою у світі, з понад 2,5 мільярдами активних користувачів у понад 190 країнах. Порівняння поширення систем Android та iOS зображено на рисунку 1.3.

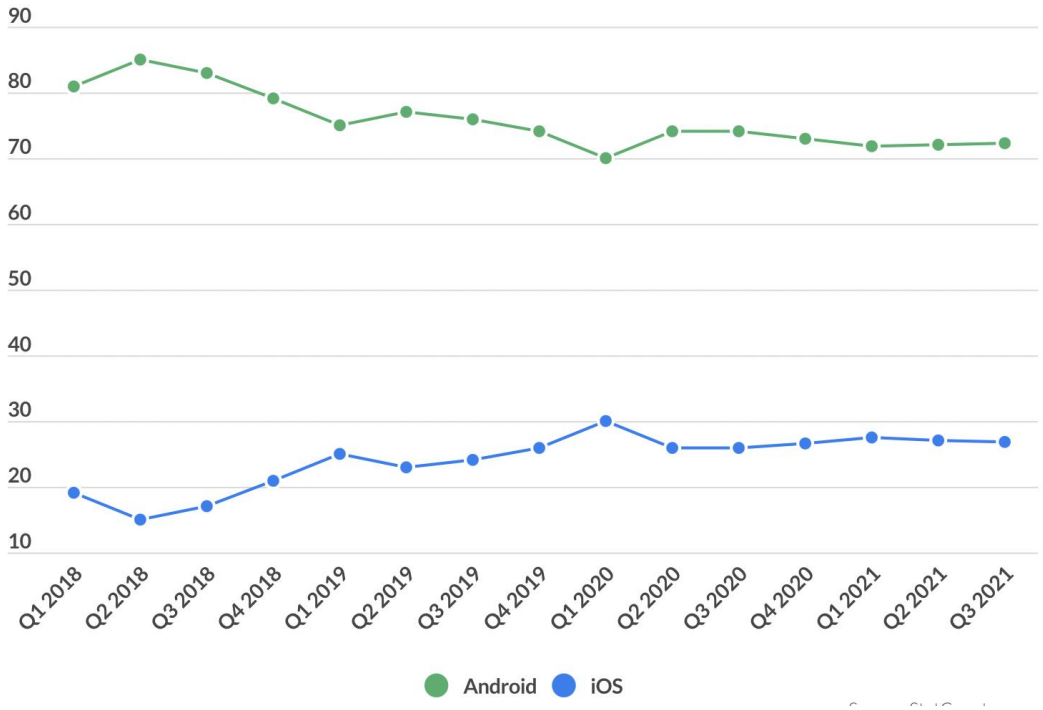

Рисунок 1.3 – Порівняння поширеності систем Android та iOS

Створений Енді Рубіном як альтернатива iPhone і Palm OS з відкритим кодом, Android швидко став улюбленою операційною системою для більшості виробників мобільних пристроїв на початку 2010-х років.

Android досить швидко стала найпопулярнішою мобільною операційною системою (ОС), досягнувши понад мільярд активних користувачів до 2014 року. У міру його зростання, зростав і контроль Google над ОС. Спочатку партнери-виробники могли налаштувати більшу частину платформи на свій смак, однак Google з кожним роком додавала все більше обов'язкових послуг та умов, що забезпечило переваги для власного набору програм.

Android є домінуючою платформою в більшості країн, хоча в Японії та Сполучених Штатах їй було важко випередити Apple. У таких країнах, як Бразилія, Індія, Індонезія, Іран та Туреччина, вона займає понад 85 відсотків ринку.

Google Play надзвичайно зріс за останнє десятиліття, досягнувши 38,6 мільярдів доларів у 2021 році. У 2021 році в магазині було понад 2,9 мільйона додатків, які було завантажено 108 мільярдів разів.

На сьогодні існує 31 версія системи Android і оскільки з кожним новим релізом змінюється кодова база та додаються нові можливості, необхідно коректно визначати мінімальну кодову базу для підтримки додатком. На рисунку 1.4 зображена дистрибуція версій системи Android, включаючи 2021 рік. Як можна бачити найстаріша версія, що досі активно підтримується це Android 5.0 під назвою Lollipop. Отже доцільним рішенням є використання саме даної версії як мінімально підтримуваної.

Окрім поширеності даної операційної системи, до переваг розробки під систему Android можна віднести більш лояльну платформу розміщення додатків. Відомо, що Apple App Store дуже вибірково ставиться до своїх опублікованих програм. Поділяючи подібні рекомендації, Google і Apple мають різний підхід до забезпечення якості. Процес затвердження в App Store може бути виснажливим і складним, але система Google підходить майже всім, якщо не порушуються її основні правила щодо вмісту. Однак Apple оцінює програми більш суворо – за наявності помилок, збоїв, невідповідностей

інтерфейсу користувача та непрацюючих посиланнь система не прийме програму.

Хоча ретельні огляди за своєю суттю не є поганими, вони означають, що розробники повинні спочатку витратити тисячі доларів, ризикуючи отримати відмову від платформи.

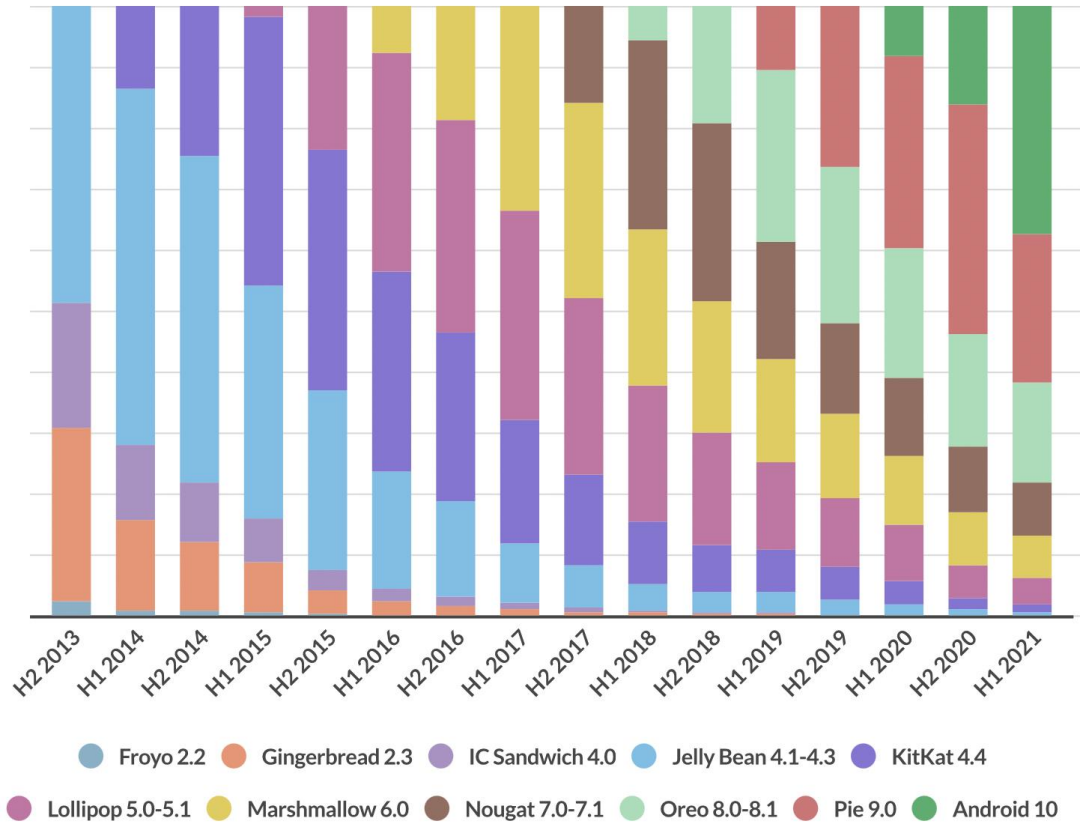

Рисунок 1.4 – Всесвітня дистрибуція версій операційної системи Android з 2013 по 2021 роки

Важливим фактором також є вимоги щодо розробки під ту чи іншу систему. Серед речей, необхідних для розробки додатків Android, апаратне забезпечення не грає великої ролі. Розробка під Android ведеться на Java та Kotlin, що робить процес кросплатформенним. Android Studio, Eclipse, IntelliJ IDEA, Fabric та багато інших інструментів розробки Android можна використовувати та завантажувати на Windows, Mac OS та Linux [3]. Для створення iOS додатків потрібно використовувати Mac або віртуальну машину.

Android має дві офіційно підтримувані мови програмування – Java і Kotlin. Перша була поширеною мовою протягом двох десятиліть і була названа 5-ю найпопулярнішою технологією в 2018 році. Java — це об'єктно-орієнтована міжплатформна мова, яка використовується скрізь, від фінтех-стартапів до ініціатив з аналізу даних. Продукти для вебу, настільних комп'ютерів, мобільних пристроїв – усі вони можуть працювати на Java.

Що стосується Kotlin, то це ще одна високо оцінена технологія. Дана мова бере на себе всю складність і багатослівність Java і робить весь процес написання програми швидшим і приємнішим.

Необхідно зазначити, що Java і Kotlin не є єдиними, хоча й офіційними, варіантами. Android Studio, наприклад, також підтримує C і C++. Обидва вони складніші, ніж Java, але можуть бути зручними в певних випадках. Крім того, якщо розглянути міжплатформенні інструменти, то мовний пул ще більший – є C# з Xamarin, JavaScript з PhoneGap та Dart з Flutter.

1.3 Загальний регламент захисту даних GDPR

Загальний регламент захисту даних (GDPR) є найсуворішим законом про конфіденційність та безпеку у світі. Незважаючи на те, що він був розроблений і прийнятий Європейським Союзом (ЄС), він накладає зобов'язання на організації в будь-якому місці, якщо вони націлені або збирають дані, пов'язані з людьми в ЄС. Постанова набула чинності 25 травня 2018 року. GDPR стягує суворі штрафи з тих, хто порушує його стандарти конфіденційності та безпеки, а штрафи сягатимуть десятків мільйонів євро [4].

З GDPR Європа демонструє свою тверду позицію щодо конфіденційності та безпеки даних у той час, коли все більше людей довіряють свої особисті дані хмарним сервісам, а порушення стають щоденним явищем. Саме регулювання є великим, далекосяжним і досить легким щодо особливостей, що робить відповідність GDPR складною перспективою, особливо для малих і середніх підприємств (МСП).

Право на приватне життя є частиною Європейської конвенції з прав людини 1950 року, де сказано: «Кожен має право на повагу до його приватного та сімейного життя, свого житла та кореспонденції». Виходячи з цього, Європейський Союз намагався забезпечити захист цього права за допомогою законодавства.

З розвитком технологій та винайденням Інтернету ЄС визнав необхідність сучасного захисту. Тому в 1995 році він прийняв Європейську директиву із захисту даних, яка встановлює мінімальні стандарти конфіденційності та безпеки даних, на основі яких кожна держава-член ґрунтує свій власний імплементаційний закон. Але Інтернет вже перетворився на сьогоднішні дані Гувера. У 1994 році в мережі з'явилася перша реклама-банер. У 2000 році більшість фінансових установ пропонували онлайн-банкінг. У 2006 році Facebook відкрився для всіх. У 2011 році користувачка Google подала до суду на компанію за сканування її електронних листів. Через два місяці після цього європейський орган із захисту даних заявив, що ЄС потребує «комплексного підходу до захисту персональних даних», і розпочалася робота над оновленням директиви 1995 року.

GDPR набув чинності в 2016 році після ухвалення Європейським парламентом, а з 25 травня 2018 року всі організації повинні були відповідати вимогам.

По-перше, якщо обробляються персональні дані громадян чи резидентів ЄС, або пропонуються товари чи послуги таким людям, тоді GDPR поширюється на компанії, навіть якщо вони не розташовані в ЄС.

По-друге, штрафи за порушення GDPR дуже високі. Існують два рівні штрафних санкцій, максимальна сума яких становить 20 мільйонів євро або 4% світового доходу (залежно від того, що вище), а також суб’єкти даних мають право вимагати компенсації за збитки.

GDPR докладно визначає низку юридичних термінів.

Персональні дані — це будь-яка інформація, яка стосується особи, яку можна прямо чи опосередковано ідентифікувати. Очевидно, що імена та

адреси електронної пошти є персональними даними. Інформація про місцезнаходження, етнічна приналежність, стать, біометричні дані, релігійні переконання, веб-файли cookie та політичні погляди також можуть бути персональними даними. Псевдонімні дані також можуть підпадати під визначення, якщо відносно легко ідентифікувати когось із них.

Обробка даних — будь-яка дія, що виконується над даними, будь то автоматизована чи ручна, включаючи збирання, запис, упорядкування, структурування, зберігання, використання, стирання тощо.

Суб’єкт даних — особа, чиї дані обробляються. Це клієнти або відвідувачі сайту чи додатку.

Контролер даних — особа, яка вирішує, чому та як будуть оброблятися персональні дані.

Обробник даних — третя сторона, яка обробляє персональні дані від імені розпорядника даних. GDPR має спеціальні правила для цих осіб та організацій. Вони можуть включати хмарні сервери, такі як Tresorit, або постачальників послуг електронної пошти, наприклад ProtonMail.

Принципи захисту даних:

— законність, справедливість та прозорість — обробка має бути законною, справедливою та прозорою для суб’єкта даних;

— обмеження цілі — дані повинні оброблятись для законних цілей, які були чітко зазначені суб’єкту даних, коли їх збирали;

— мінімізація даних — необхідно збирати та обробляти лише стільки даних, скільки абсолютно необхідно для зазначених цілей;

— точність — необхідно підтримувати точні та актуальні особисті дані;

— обмеження зберігання — можна зберігати особисті дані лише стільки, скільки необхідно для зазначеної мети;

— цілісність і конфіденційність — обробка повинна здійснюватися таким чином, щоб забезпечити належну безпеку, цілісність і конфіденційність (наприклад, за допомогою шифрування);

— підзвітність — контролер даних несе відповідальність за те, щоб продемонструвати відповідність GDPR всім цим принципам.

Дані повинні оброблятись із вживанням «відповідних технічних та організаційних заходів».

Усе, що відбувається в організації, має «за проектом і за замовчуванням» враховувати захист даних. Практично кажучи, це означає, що принципи захисту даних повинні враховувати при розробці будь-якого нового продукту або діяльності. GDPR охоплює цей принцип у статті 25.

Наприклад, при запуску нового додатку треба подумати про те, які персональні дані програма може збирати від користувачів, потім розглянути способи мінімізації обсягу даних і способи їх захисту за допомогою новітніх технологій.

У статті 6 перелічено випадки, коли обробка даних про особу є законною:

— суб’єкт даних дав конкретну, недвозначну згоду на обробку даних (наприклад, вони увійшли до списку маркетингової розсилки);

— обробка необхідна для виконання або підготовки до укладання договору, стороною якого є суб’єкт даних;

— потрібно обробити дані особи, щоб виконати юридичні зобов’язання;

— потрібно обробити дані, щоб врятувати чиєсь життя;

— обробка необхідна для виконання завдання в суспільних інтересах або для виконання якоїсь службової функції;

— за наявності законного інтересу обробки персональних даних певної людини.

Це найбільш гнучка законна основа, хоча «основні права та свободи суб’єкта даних» завжди мають перевагу, особливо якщо це дані дитини.

Після того, як були визначені законні підстави для обробки даних, потрібно задокументувати цю підставу та повідомити суб’єкта даних. І якщо пізніше буде вирішено змінити обґрунтування, потрібно мати поважну причину, задокументувати цю причину та повідомити суб’єкта даних.

Існують суворі нові правила щодо того, що є згодою суб’єкта даних на обробку його інформації:

— згода має бути «вільно наданою, конкретною, поінформованою та недвозначною»;

— запити про згоду мають «чітко відрізнятися від інших питань» і подані «ясною та зрозумілою мовою»;

— суб’єкти даних можуть відкликати надану раніше згоду, коли захочуть, і ви повинні поважати їхнє рішення, оскільки не можна просто змінити правову основу обробки на одне з інших обґрунтувань;

— діти до 13 років можуть давати згоду лише з дозволу батьків;

— необхідно зберігати документальне підтвердження згоди.

З наведених даних можна зробити висновок, що дотримання закону GDRP є необхідністю при розробці бібліотеки для збору аналітичних даних, оскільки ця інформація підпадає під категорію особистих даних і повинна оброблятись з достатньою обережністю.

1.4 Огляд існуючих бібліотек для збору аналітичних даних

На сьогодні можна виділити дві найбільш популярних бібліотеки для збору та відправки аналітичних даних: Firebase Analytics та Segment Analytics.

1.4.1 Firebase Analytics

Firebase Analytics — це безкоштовне програмне рішення для аналізу додатків, яке надає уявлення про використання додатків і залучення користувачів.

В основі Firebase лежить Google Analytics — безкоштовне та необмежене аналітичне рішення. Analytics інтегрує всі функції Firebase і надає необмежену кількість звітів для 500 унікальних подій, які можна визначити за допомогою Firebase SDK.

SDK автоматично фіксує ряд подій і властивостей користувача, а також дозволяє визначати власні користувацькі події для вимірювання речей, які мають унікальне значення для бізнесу. Після отримання даних вони доступні на інформаційній панелі через консоль Firebase. Ця інформаційна панель надає детальну інформацію про дані — такі як активні користувачі та демографічні показники.

Приклади подій, які Google Аналітика для Firebase збирає за замовчуванням:

— кількість користувачів та сеансів;
— тривалість сеансу;
— операційні системи;
— моделі пристроїв;
— місцезнаходження;
— перші запуски;
— відкриття додатків;
— оновлення програм;
— придбання у додатку.

Firebase Analytics не реєструє події, параметри подій та властивості користувачів, які перевищують обмеження, зазначені у таблиці 1.1 [5].

З наведених даних можна зробити висновок, що разом з тим, що Firebase Analytics надається умовно безкоштовно, дана бібліотека накладає велику кількість обмежень, що може суттєво вплинути на кінцевий результат аналізу данних.

Таблиця 1.1 — Обмеження бібліотеки Firebase Analytics

Об’єкт реєстрації	Обмеження
Окремі події	500 на екземпляр додатку, не враховуються події, що реєструються автоматично, такі як first_open та in_app_purchase
Довжина імені події	40 символів
Кількість параметрів події	25
Довжина імені параметра події	40 символів
Довжина значення параметра події	100 символів
Властивості користувача	25 на екземпляр додатку
Довжина імені властивості користувача	24 символи
Довжина значення властивості користувача	36 символів
Вік події (час завантаження)	Події, завантажені більш ніж через 2 дні після реєстрації не експортуються

Firebase Analytics не має детальної документації, а код бібліотеки обфускований. Обфускація коду — це процес модифікації виконуваного файлу, щоб він більше не був корисним для хакера, але залишався повністю функціональним. Хоча процес може змінювати фактичні інструкції методу або метадані, він не змінює результат програми.

Бібліотека надає примітивні можливості щодо роботи з нею. Як і зазвичай це виклик методу з назвою події та параметрами, при чому параметри обмежені типом даних Bundle системи Android, що значно знижує зручність роботи та накладає певні обмеження. Серед додаткових можливостей присутні задання ідентифікатора користувача та параметрів, пов’язаних з ним, а також

можливість задання згоди користувача на відслідковування аналітичних даних.

Інтерфейс роботи з бібліотекою зображений на рисунку 1.5.

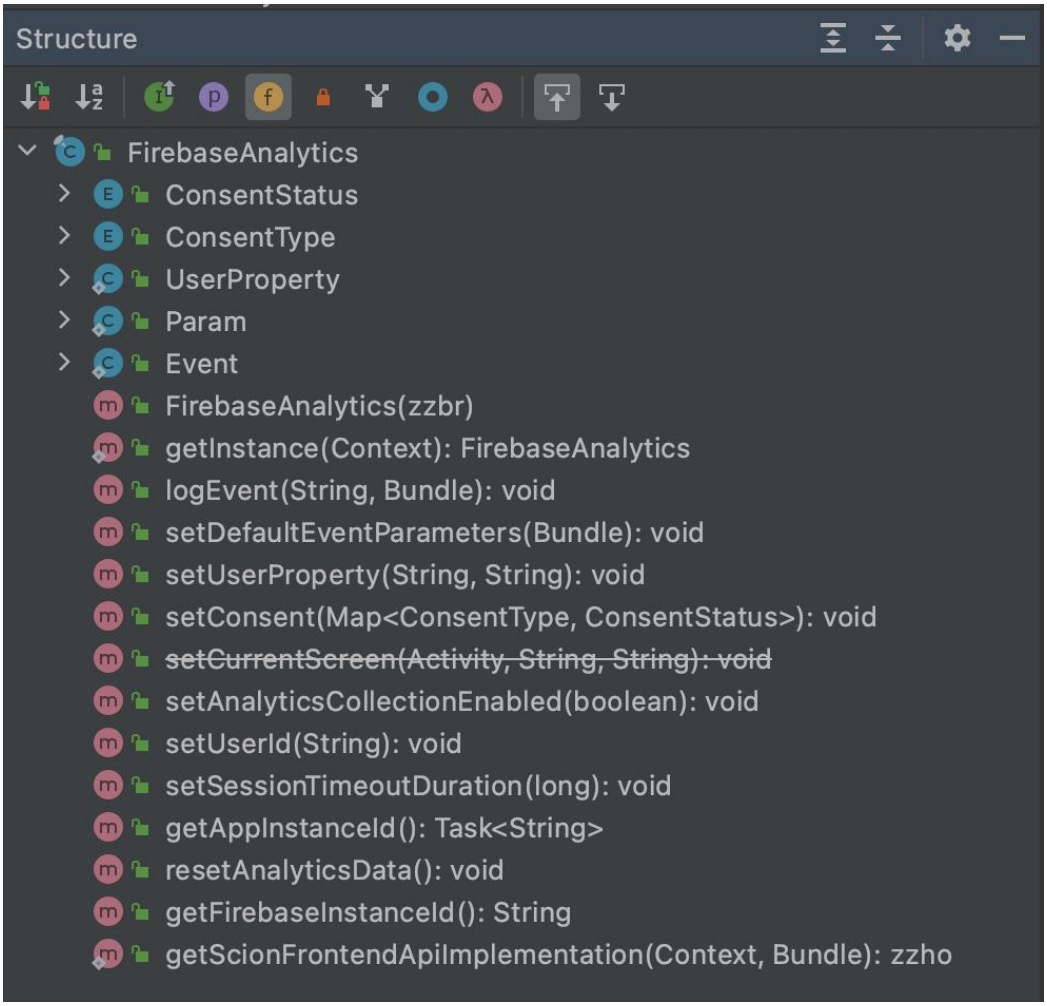

Рисунок 1.5 – Інтерфейс роботи з бібліотекою Firebase Analytics

1.4.2 Segment Analytics

Segment Analytics — це платформа даних клієнтів (CDP), що означає, що послуги які надає ресурс спрощують збір і використання даних від користувачів на різних цифрових ресурсах (веб-сайтах, програмах тощо). За допомогою Segment можна збирати, перетворювати, надсилати й архівувати дані клієнтів.

Бібліотеки Segment генерують і надсилають повідомлення до певного API, призначеного для відстеження, у форматі JSON. Бібліотеки надають

стандартну структуру для основних викликів API, а також рекомендовану структуру JSON (також відому як "Spec", тип схеми), яка допомагає підтримувати узгодженість найважливіших частин даних, водночас забезпечуючи велику гнучкість у тому, яка інформація збирається, і де.

Для найпростішого повідомлення потрібен лише ідентифікатор користувача або анонімний ідентифікатор. Всі інші поля є необов’язковими, щоб забезпечити максимальну гнучкість. Однак звичайне повідомлення має три основні частини: загальні поля, об’єкт «контекст» і властивості (англійською «properties»), якщо це подія або ознаки (англійською «traits») якщо це об’єкт.

Загальні поля включають інформацію про те, як був створений виклик, зокрема штамп часу, назву та версію бібліотеки. Поля в об’єкті контексту зазвичай генеруються бібліотекою і містять інформацію про середовище, в якому було створено виклик: шлях до сторінки, агент користувача, ОС, параметри локалі тощо.

Іншою поширеною частиною повідомлення Segment є об’єкт інтеграції, який можна використовувати для явного фільтрування, на які кінцеві точки переадресовується виклик. Однак цей об’єкт є необов’язковим і часто опускається на користь параметрів фільтрації, не заснованих на коді.

Segment надає кілька типів джерел, які можна використовувати для збору даних і які можна вибрати відповідно до потреб додатка чи сайту. Для веб-сайтів можна використовувати бібліотеку, яка завантажується на сторінку, щоб створити повідомлення. Якщо розробляється мобільний додаток, можна використати одну з мобільних бібліотек. Схема роботи сервісів Segment зображена на рисунку 1.6.

Analytics для Android підтримує лише будь-який пристрій Android із версією API 14 (Android 4.0) або новіше, включаючи пристрої Amazon Fire.

Однією з найважливіших частин будь-якої аналітичної платформи є здатність послідовно та точно ідентифікувати користувачів. Для цього платформа повинна призначити та зберегти певну форму ідентифікації на

пристрої, щоб можна було ефективно аналізувати дії користувача. Це особливо важливо для аналізу конверсій і утримання.

Звичайно, пакет Analytics SDK потребує унікального ідентифікатора для кожного користувача. Під час першого запуску програми Android, яка використовує Segment, Segment SDK генерує унікальний ідентифікатор UUID (Universally Unique Identifier) і зберігає його у постійній пам’яті пристрою. Він використовується як anonymousId і залишається незмінним для користувача на пристрої. Щоб створити нового користувача на тому ж пристрої, необхідно викликати скидання на клієнті Analytics [6].

Segment SDK також збирає рекламний ідентифікатор, наданий сервісами Google Play. Це ідентифікатор, який слід використовувати в рекламних цілях. Це значення встановлено на context.device.advertisingId. Ідентифікатор Android встановлено як context.device.id. Деякі пункти призначення покладаються на те, що це поле є ідентифікатором Android.

Бібліотека ставить в чергу виклики API і завантажує їх пакетами. Це обмежує кількість здійснених мережевих викликів і допомагає заощадити заряд акумулятора на пристрої користувача.

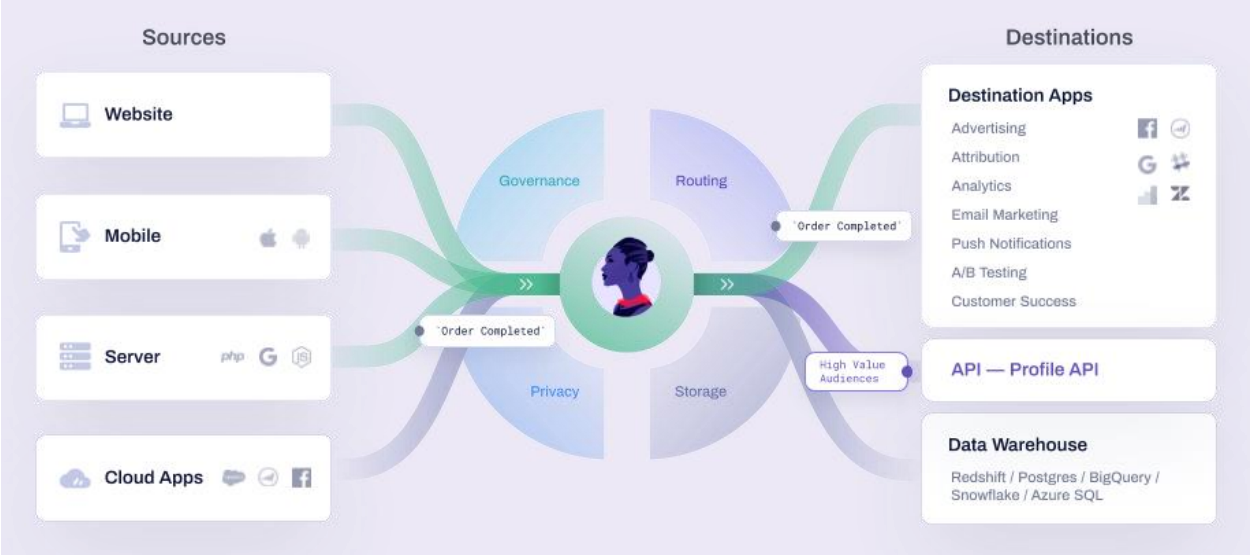

Рисунок 1.6 – Схема роботи сервісів Segment

Коли подія надсилається, бібліотека зберігає її на внутрішню пам’ять. Коли розмір черги досягає максимального розміру, який був вказаний

клієнтським кодом (20 за замовчуванням), бібліотека очищає чергу та завантажує події одним пакетом. Оскільки дані зберігаються миттєво, вони не втрачаються, навіть якщо програму буде знищено або операційна система аварійно завершує роботу.

Поведінка черги може відрізнятися для місць призначення в режимі пристрою. Наприклад, SDK Mixpanel ставить події в чергу, а потім очищає їх, лише коли програма переходить у фоновий режим.

Segment Analytics дозволяє відстежувати обмежену кількість властивостей подій. Сторінка з інформацією про подію може показувати лише перші 300 властивостей. Після того, як буде досягнуте обмеження в 300 властивостей, майбутні властивості все ще відстежуються та надсилаються до місць призначення, але вони не відображатимуться на сторінці деталей події. Це обмеження включає вкладені властивості в об'єкт властивостей події.

Серед функціональних властивостей доступно відслідковування зміни екранів при зміні Activity додатку. На даний момент це є застарілим підходом до побудови додатків, тому дане рішення не є актуальним на сьогодні.

Також Segment Analytics автоматично збирає доволі широкий набір параметрів, серед яких інформація про додаток (номер збірки, назва, версія), інформація про пристрій (рекламний ідентифікатор користувача, Android ID, виробник, модель), IP – адреса, локаль, часовий пояс, характеристики екрану, а також статус бездротових модулів (Wi-Fi, Bluetooth, мобільний зв'язок).

З іншого боку, інтерфейс для роботи з бібліотекою є ідентичним з бібліотекою Firebase Analytics. Структура головного класу зображена на рисунку 1.7. Візуально інтерфейс роботи з бібліотекою здається більш обширним, але причиною цього є велика кількість перевантажених методів. Дані методи виконують аналогічні дії, приймаючи на вхід різні списки параметрів.

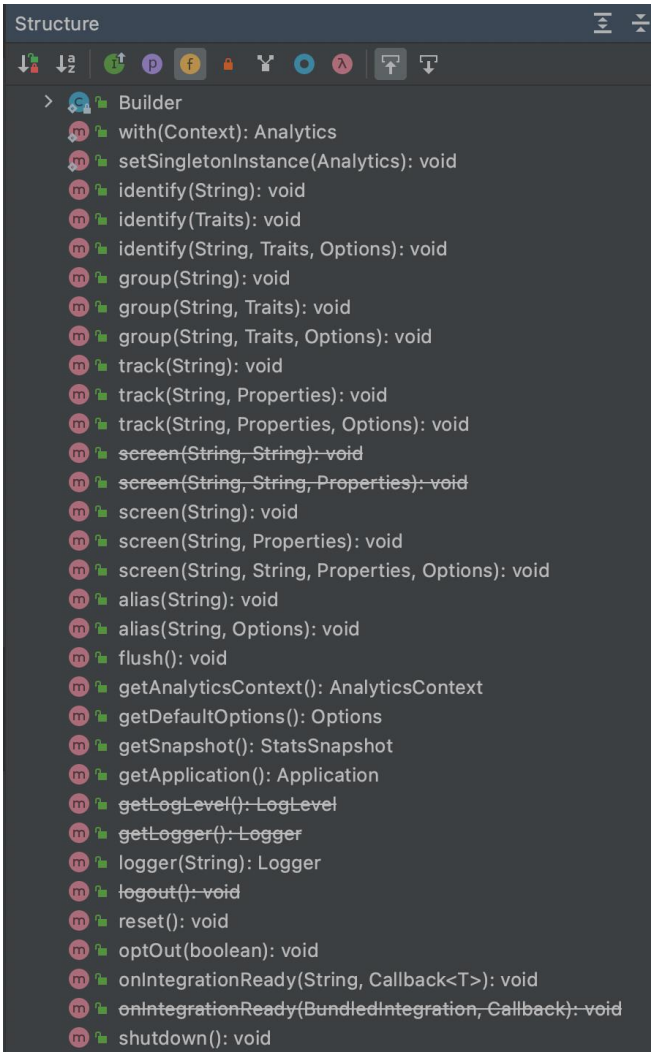

Рисунок 1.7 – Інтерфейс роботи з бібліотекою Segment Analytics

Отже, обидві бібліотеки надають схожі функціональні можливості для роботи, але Segment Analytics є більш універсальним рішенням хоча і потребує постійну оплату за свої послуги. В кінцевому рахунку, обидві бібліотеки надають лише базові функціональні можливості для збору аналітичних даних і потребують велику кількість шаблонного коду, що повторюватиметься відповідно до кількості даних, які необхідно зібрати. Тому обидва рішення не відповідають вимогам поставленим щодо проекту який розробляється.

2 МЕТОДИ ТА ТЕХНОЛОГІЇ ДЛЯ ЗБОРУ Й ОБРОБКИ АНАЛІТИЧНИХ ДАНИХ

2.1 Розробка вдосконаленого методу збору та обробки аналітичних даних

Головною ідеєю методу, що розробляється, є спрощення реалізації процесу збору та обробки аналітичних даних у коді клієнтського додатку. Ці дані формуються у вигляді події з певним набором параметрів. Відповідно, для формування події необхідні назва події, ключі для кожного з параметрів та самі дані. Оскільки в об’єктно-орієнтованому коді за виконання будь-яких дій відповідають методи, можна зробити висновок, що саме вони будуть джерелом аналітичних подій [7]. У цьому випадку можна автоматизувати даний процес, певним чином позначаючи необхідні методи, використовуючи сигнатуру метода, а саме його назву та назви параметрів, як вихідні дані для генерування події.

Оскільки у системі Android у кожного додатку наявне лише одне активне вікно на якому знаходиться користувач, воно може виступати у вигляді контексту, у рамках якого виконуються дії. Тому доцільним є збереження назви екрану та його подальше використання у згенерованих подіях, а також відправка відповідної події про зміну екрану.

Контекст екрану може розширятись не лише його назвою, а й певним набором параметрів, який є актуальним для нього. Через це постає необхідність у створенні додаткового сховища параметрів, що зберігало б дані, які відносяться до поточного екрану. Завдяки відслідковуванню зміни екранів, надається можливість очищати це сховище на кожний перехід.

Так як усі події повинні бути унікальними, немає сенсу у обробці даних, що не є релевантними для конкретної події. Для вирішення даного питання можна скористатись динамічною конфігурацією додатку, що завантажуватиметься віддалено. Таким чином, файл конфігурації вказуватиме набір параметрів, який вимагається для події, що дозволить відфільтровувати

дані в коді та цим самим економити оперативну пам’ять пристрою та кількість інтернет трафіку, що надсилається.

Дані підходи дозволяють автоматизувати процес збору та обробки аналітичних даних наскільки це можливо, враховуючи необхідність абстрагування від клієнтського коду, в рамках якого буде використовуватись бібліотека. Візуалізація методу зображена на рисунку 2.1. В подальших підрозділах детально описуються підходи та технології, що необхідно застосувати для реалізації даного методу.

Рисунок 2.1 – Візуалізація роботи вдосконаленого методу збору й обробки аналітичних даних

2.2 Автоматизації генерування аналітичних подій з використанням аспектно-орієнтованого програмування

Аспектно-орієнтоване програмування (АОП) є однією з концепцій програмування, яка є подальшим розвитком процедурного та об'єктно-орієнтованого програмування (ООП). Ця методологія покликана знизити час, вартість та складність розробки програмного забезпечення. У сучасному ПЗ, як правило, можна виділити певні частини, або аспекти, що відповідають за ту чи іншу функціональність, реалізація якої розосереджена по коду програми, але складається з подібних шматків коду. За оцінками фахівців, близько 70% часу в проектах витрачається на супровід та внесення змін до готового програмного коду. Тому у найближчій перспективі роль АОП та подібних трансформаційних підходів стає досить важливою.

Сучасні програмні системи мають дуже високий рівень складності: один розробник практично не в змозі охопити всі деталі системи. Складність програмних систем обумовлена кількома причинами: складністю реальної предметної області, з якої виходить замовлення на розробку; необхідністю забезпечення достатньої гнучкості програми; незадовільними методами опису поведінки великих систем.

АОП передбачає наявність мовних засобів, що дозволяють виділяти наскрізну функціональність в окремі модулі. Це дозволяє спрощувати роботу (налагодження, модифікування, документування, тощо) з компонентами програмної системи та знижувати складність системи загалом. Під модулем (компонентом) розуміється деяка чітко виражена структурна одиниця програми процедура, функція, метод, клас або пакет.

При розробці програмної системи з використанням існуючих методологій програмування наскрізна функціональність буде включена у всі модулі, в результаті система буде складною у проектуванні, розумінні, реалізації та підтримці.

Програмна система створюється як наслідок обробки безлічі вимог. Можна явно виділити з цієї множини вимоги до логіки конкретного модуля та загальносистемні вимоги.

Сучасні технології розробки програмного забезпечення надають зручні засоби для виділення логіки функціонування програми в окремі модулі, але жодна з них не пропонує зручного способу локалізації в окремі модулі функціональності, яка повинна поширюватися на всю систему.

Аспектно-орієнтований підхід розглядає програмну систему як набір модулів, кожен із яких відображає певний аспект — мета, особливість функціонування системи. Набір модулів, що утворюють програму, залежить від вимог до програми, особливостей її предметної галузі. При проектуванні програмної системи розробник вибирає модулі так, щоб кожен з них реалізовував певну функціональну вимогу до системи. Однак реалізація деяких вимог до програми часто не може бути локалізована в окремому модулі в рамках процедурного або об'єктно-орієнтованого підходу. В результаті код, що відображає такі аспекти функціонування системи, буде зустрічатися в різних модулях. Традиційні парадигми програмування використовують при проектуванні програми функціональну декомпозицію і дозволяють локалізувати наскрізну функціональність окремих модулях. Необхідність реалізації наскрізної функціональності наявними засобами веде до того, що деякий компонент містить код, що відображає безліч ортогональних вимог до системи. Це робить такий модуль вузькоспеціалізованим, погіршує можливості його повторного використання та в деяких випадках призводить до дублювання коду. У свою чергу, це викликає підвищення ймовірності внесення помилок, збільшення часу налагодження, знижує якість програми та значною мірою ускладнює її супровід. Аспектно-орієнтований підхід дозволяє уникнути описаних проблем та покращити загальний дизайн системи, забезпечуючи можливість локалізації наскрізної функціональності у спеціальних модулях – аспектах [8].

Таким чином за допомогою аспектно-орієнтованого програмування, зручно виділити в окремий модуль логіку щодо збору даних, щоб даний код міг бути застосований в будь-якому місці додатку без його повторного написання. Оскільки операційна система Android працює на віртуальній машині JVM, практично єдиним варіантом реалізації АОП є бібліотека AspectJ, що розроблена для мови Java [9].

AspectJ сам по собі досить простий у використанні. Однак у контексті програми для Android все починає ускладнюватися. AspectJ працює за принципом модифікації зкомпільованого Java байт коду. Android же, у свою чергу, працює на власній реалізації байт коду під назвою DEX. DEX код незважаючи на те, що базується на Java є несумісним з AspectJ. Але при вивченні послідовності збірки Android додатку (рисунок 2.2), можна побачити, що у списку є команда app:compileDebugJavaWithJavac, що відповідає за компіляцію Java байт коду. З цього можна зробити висновок, що на певному етапі компіляції існує AspectJ сумісний код. Ціль полягає у додаванні задачі, яка б виконувалась саме на цьому моменті і дозволяла подальшу компіляцію додатку.

Існуючі рішення написані неякісно і не дозволяють їх використання у важливих додатках. Також важливою проблемою є той факт, що дані рішення реалізовані виключно для мови Java. У випадку ж використання Kotlin, дані фрагменти коду відмовляються працювати. Це відбувається тому, що з поєднанням Java/Kotlin, Gradle повинен скомпілювати не тільки кожну задачу окремим кроком, але є й наступні завдання, які вже чекатимуть на виконання після компіляції. Тепер їм потрібно дочекатися завершення завдання, яке додає AspectJ код.

2.3 Метод ініціалізації бібліотеки та керування її станом

Для ініціалізації бібліотеки необхідно передавати певний набір параметрів, при чому деякі параметри можуть бути відсутні. Для подібних цілей було створено шаблон проектування Builder.

Мета шаблону проектування Builder полягає в тому, щоб відокремити конструкцію складного об’єкта від його представлення. Цей шаблон надає низку методів, які забезпечують прозорість для користувача класу, щоб краще зрозуміти, що відбувається під капотом.

Шаблон конструктора зберігає значення логіки та конфігурації за замовчуванням, необхідні для створення об’єкта в класі конструктора. Це дозволяє створювати об’єкти з мінімальними даними конфігурації і без необхідності знати значення за замовчуванням, які будуть використовуватися для створення об’єкта.

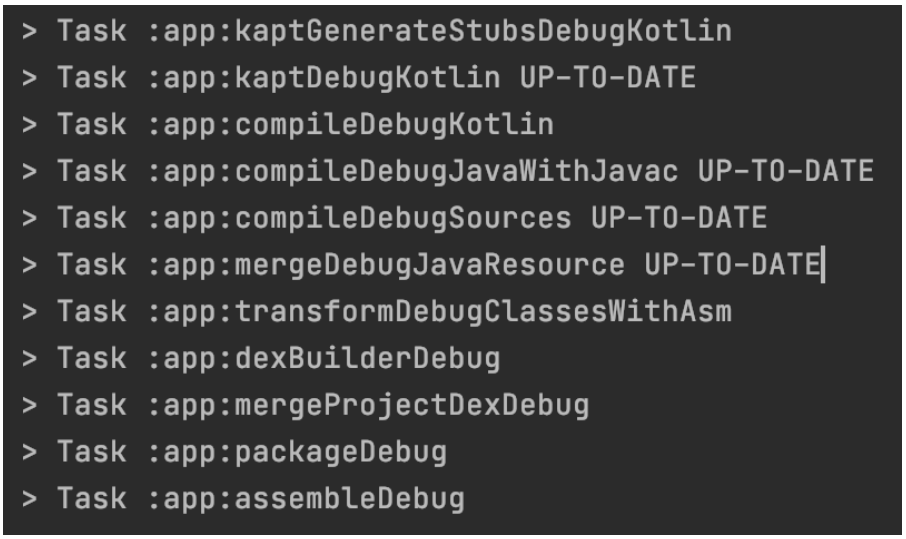

Рисунок 2.2 – Список задач завершення компіляції Android додатку

Цей шаблон полегшує зміну значень конфігурації за замовчуванням, які використовуються для створення об’єкта, і зміну класу, з якого створюються екземпляри.

Його доцільно використовувати коли для створення об’єкта потрібен складний процес конфігурації, і необхідно, щоб значення конфігурації за замовчуванням не розповсюджувалися по всій програмі. Таким же чином цей шаблон вирішує проблему наявності багатьох необов’язкових параметрів.

Шаблон конструктора вирішує проблему, вводячи посередника, який називається конструктором, між компонентом і об’єктом, з яким він повинен працювати [9].

У першій операції конструктору надається елемент даних, який замінює одне із значень за замовчуванням (якщо воно присутнє), що використовуються для створення об’єкта.

У другій операції відбувається створення об’єкту. Це сигналізує, що нових даних не буде і що об’єкт має бути створений із використанням значень даних, які він отримав до цього часу, разом із значеннями за замовчуванням для елементів даних, які не були вказані.

У третій операції конструктор створює об’єкт і повертає його (рисунок 2.3).

Незважаючи на невелику кількість параметрів, необхідних для ініціалізації бібліотеки потрібно усвідомлювати можливу необхідність розширення функціональних можливостей бібліотеки і, відповідно, кількості вхідних параметрів.

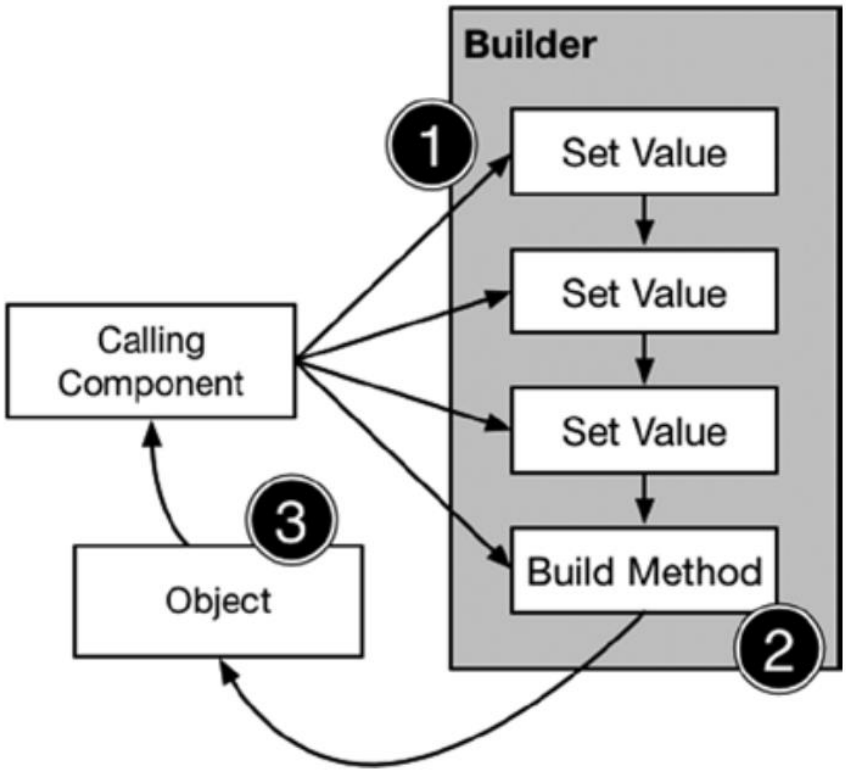

Рисунок 2.3 – Принцип роботи шаблону Builder

Таким чином, використання даного шаблону є повністю виправданим, адже при всіх перевагах, він ніяким чином не ускладнює роботу бібліотеки.

Важливою частиною ініціалізації додатку є його конфігурація, що викачується при запуску. Стан бібліотеки формуватиметься в залежності від даного конфігураційного файлу та стану згоди на обробку персональних даних. Таким чином бібліотека може перебувати в трьох станах:

— ввімкнена (Enabled);

— вимкнена (Disabled);

— не ініціалізована (Not initialized).

Перехід в перші два стани відбудеться за умови успішного виконання або не виконання обох умов відповідно, третій же стан використовується за замовчуванням та зберігається у випадку неможливості завантаження конфігураційного файлу.

Файл конфігурації містить:

— перемикач стану бібліотеки для віддаленого вимкнення за необхідності;

— кількість подій при досягненні якої необхідно проводити їх відправку та очищення;

— період часу в секундах через який будуть відсилатись накопичені події;

— плани для подій, що включають в себе набір параметрів для тих чи інших подій, що дозволяє проводити фільтрацію і економити пам’ять пристрою та інтернет трафік;

— дані для конфігурації екрану згоди щодо обробки персональних даних.

2.4 Підхід до асинхронної роботи з використанням Kotlin Coroutines

Kotlin Coroutines представляють новий стиль реалізації паралелізму, який можна використовувати на Android для спрощення написання асинхронного коду. Хоча корутини є новими для Kotlin, концепція корутин

існувала з самого початку мов програмування. Першою мовою, в якій проводилось їх дослідження була Simula в 1967 році [11].

За останні кілька років корутини зросли в популярності, і тепер вони включені в багато популярних мов програмування, таких як Javascript, C#, Python, Ruby і Go. Корутини Kotlin засновані на загальновизнаних концепціях, які використовувалися для створення великих додатків.

На Android корутини підходят для рішенням двох основних проблем:

— виконання довго виконуваних завдань (завдання, які займають занадто багато часу та блокують основний потік);

— main-safety дозволяє гарантувати, що будь-яку функцію призупинення можна викликати з основного потоку.

Отримання веб-сторінки або взаємодія з API включають створення мережевого запиту. Аналогічно, читання з бази даних або завантаження образу з диска передбачає читання файлу. Такі речі і є довготривалими завданнями.

Може бути важко зрозуміти, наскільки швидко сучасний телефон виконує код порівняно з мережевим запитом. На Pixel 2 один цикл процесора займає трохи менше 0,0000000004 секунд. Таким чином за повільний мережевий запит, центральний процесор може виконати понад мільярд циклів.

На Android кожна програма має основний потік, який відповідає за обробку інтерфейсу користувача (наприклад, відмальовування відображень) і координацію взаємодії користувачів. Якщо на цьому потоці виконується занадто багато роботи, програма зависає або сповільнюється, що призводить до небажаної взаємодії з користувачем. Будь-яке довготривале завдання слід виконувати без блокування основного потоку, щоб програма не відображала те, що називається «джанк», наприклад, заморожену анімацію, або повільно реагувала на події дотику.

Щоб виконати мережевий запит поза основним потоком, поширеним шаблоном є зворотні виклики. Зворотні виклики забезпечують обробник для

бібліотеки, який вона може використовувати для зворотного виклику коду в майбутньому.

Корутини будуються на основі звичайних функцій, додаючи дві нові операції. На додаток до виклику (invoke) і повернення (return), корутини додають призупинення (suspend) та відновлення (resume). Suspend призупиняє виконання поточної корутини, зберігаючи всі локальні змінні. Resume продовжує призупинену корутину з того місця, де її було призупинено.

Цю функціональність Kotlin додає за допомогою ключового слова suspend до функції. Функції призупинення можуть бути викликані лише з інших функцій призупинення або за допомогою конструктора корутин, наприклад launch, для запуску нової корутини. Таким чином використання suspend та resume разом дозволяє замінити зворотні виклики.

Щоразу, коли корутина призупиняється, поточний фрейм стека (місце, яке Kotlin використовує, щоб відстежувати запущену на даний момент функцію та її змінні) копіюється та зберігається на потім. Коли вона відновиться, фрейм стека копіюється з того місця, де його було збережено, і знову розпочинає свою роботу. У середині анімації — коли всі корутини основного потоку призупинено, основний потік може вільно оновлювати екран і обробляти події користувача.

У корутинах Kotlin добре написані функції призупинення завжди безпечні для виклику з основного потоку. Незалежно від того, що вони роблять, вони завжди повинні дозволяти будь-якому потоку викликати їх.

Є багато речей, які виконують в програмах для Android, що є надто повільними для роботи в основному потоці: мережеві запити, парсинг JSON, читання чи запис бази даних або навіть просто перебирання великих списків. Будь-яка з перечислених задач може працювати досить повільно, щоб викликати видимий для користувача «джанк» і саме тому повинна бути запущена з фонового потоку.

Використання призупинення не вказує Kotlin запускати функцію у фоновому потокі. Варто чітко і часто вказувати, що корутини будуть

виконуватися в основному потоці. Гарною ідеєю є використання Dispatchers.Main.immediate під час запуску корутини у відповідь на подію інтерфейсу користувача. Таким чином, якщо не виконуватиметься довготривале завдання, яке вимагає main-safety, результат може бути доступним у наступному кадрі для користувача [12].

Щоб зробити функцію, яка працює надто повільно для основного потоку, безпечною для нього, можна вказати корутинам Kotlin виконувати роботу або з диспетчером за замовчуванням, або з диспетчером IO. У Kotlin всі корутини повинні виконуватися в диспетчері — навіть якщо вони запущені в основному потоці. Корутини можуть призупинити себе, а диспетчер — це той, хто знає, як їх відновити.

Щоб визначити, де повинні виконуватися корутини, Kotlin надає три диспетчери, які можна використовувати для диспетчеризації потоків:

— Dispatchers.Main — основний потік на Android для взаємодії з інтерфейсом користувача та виконання легкої роботи;

— Dispatchers.IO — оптимізований для дискового та мережевого вводу-виводу поза основним потоком;

— Dispatchers.Default — оптимізований для інтенсивної роботи процесора поза основним потоком.

Очевидно, що підхід з використанням корутин значно полегшує роботу з фоновими потоками в додатках, що використовують мережеві виклики та роботу з файлами.

2.5 Аналіз реалізації зміни екранів у сучасних Android додатках

Більшість програм на Android містять кілька екранів, по яких користувач переміщається за допомогою жестів, натискання кнопок і вибору меню. До впровадження компонентів Android Jetpack реалізація навігації в програмі в основному була ручним процесом кодування без легкого способу перегляду та

організації потенційно складних шляхів навігації. Ця умова була значно покращена завдяки введенню Android Jetpack Navigation Component.

В основному, навігація відноситься до взаємодій, які дозволяють користувачам переміщатися між різними частинами вмісту у додатку. Іншими словами, навігація по програмі, переміщення з одного екрана на інший є абсолютно фундаментальною частиною розробки Android. Раніше це виконувалось за допомогою намірів (Intent) або фрагментних транзакцій, але на сьогодні дані підходи є застарілими та не використовуються. Навігаційний компонент Android Jetpack допомагає реалізувати навігацію, від простих натискань кнопок до більш складних шаблонів, таких як панелі програм і панель навігації. Точніше, компонент навігації — це набір бібліотек, плагінів та інструментів, які спрощують навігацію Android. Крім спрощення налаштування звичайних шаблонів, таких як нижня навігація, компонент обробляє стек, транзакції фрагментів, передачу аргументів, анімацію на основі навігації та глибоке посилання. Найголовніше, він може зібрати всю цю навігаційну інформацію та помістити її в одне візуалізоване розташування додатка за допомогою графіка навігації. Фактично, компонент Navigation включає три основні частини: графік навігації, NavHost і NavController [13].

NavController дозволяє зареєструвати власний лістенер, який викликатиметься на зміну екрану, при цьому передаючи до нього такі дані, як його назва та ідентифікатор. Це дозволить успішно відслідковувати зміну екранів та визначати які екрани мають ігноруватись за отриманим ідентифікатором.

Система Android також надає стандартний компонент під назвою ViewPager, який реалізує графічний інтерфейс сторінок, між якими можна переміщатись [14]. Дані сторінки можна класифікувати як підвид екрану, що в свою чергу також потребує способу відслідковування. Для цього у бібліотеці ViewPager наявний відповідний лістенер — OnPageChangeListener. Однак даний лістенер викликається після виклику методу onViewCreated() життєвого

циклу фрагменту, що викликає певні труднощі. Відповідно до архітектури MVVM (Model-View-ViewModel) отримання даних фрагментом відбувається за допомогою LiveData, що являється реалізацією шаблону Observer, який викликається відповідно до життєвого циклу підписаного фрагменту. Активним вважається стан після виклику методу onStart() та до виклику onStop() (рисунок 2.4) [15].

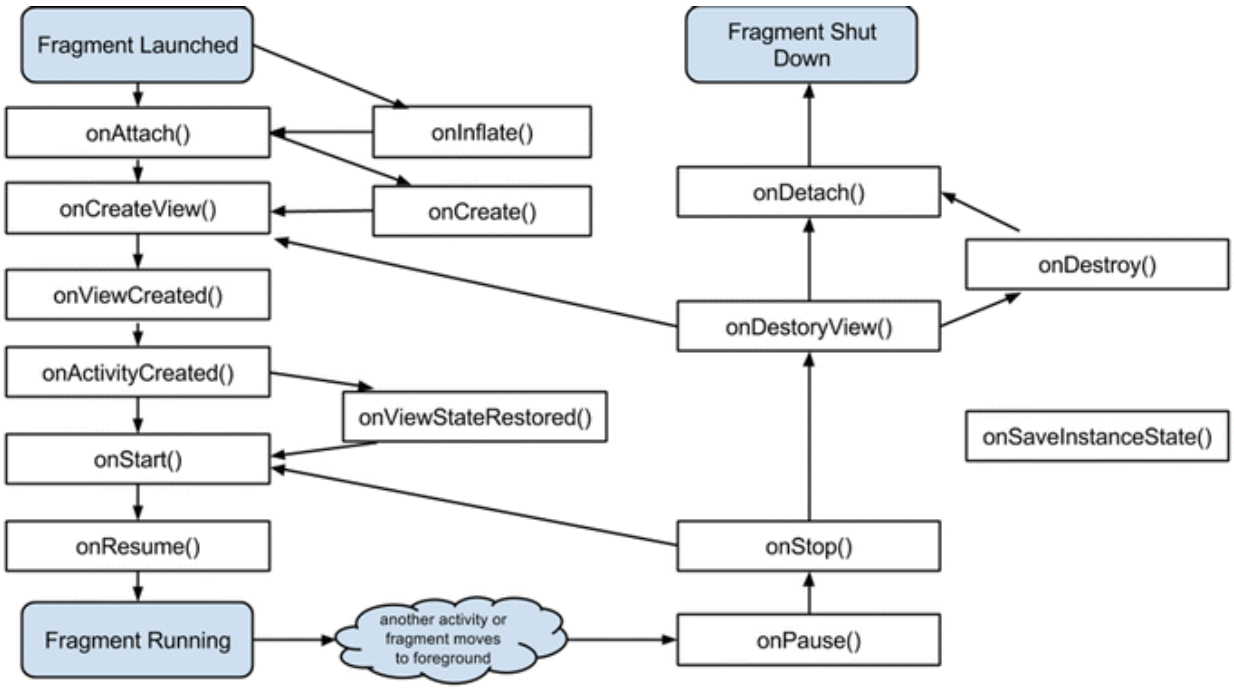

Рисунок 2.4 – Життєвий цикл фрагмента у системі Android

Таким чином, постає необхідність у написанні власного варіанту відслідковування зміни сторінок ViewPager, яка б викликалась в активний момент життєвого циклу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ БІБЛІОТЕКИ

3.1 Написання Gradle плагіна для підтримки АОП

Для написання власного Gradle плагіна існує два можливих варіанти: використання Groovy DSL або ж Kotlin KTS [16]. У ході роботи було вирішено використання останнього, через можливості перевірки безпеки типів і швидшого часу компіляції.

Для подібних задач існує «Конвеєр маніпуляцій з байт-кодом Android» (Android bytecode manipulation pipeline). Цей механізм спеціально призначений для інструментів маніпуляції байт-кодом. З його використанням можна відмовитися від складної перебудови залежностей завдань Gradle і просто зареєструвати власну логіку як маніпулятор байт-кодом. Дана логіка почнеться точно в потрібний час і не впливатиме на залежні завдання.

Це рішення, може працювати як для Java, так і для Kotlin, і не порушувати механізми кешу збірки Gradle.

Отже, необхідно написати плагін Gradle, який виконує влаштування АОП за допомогою техніки, рекомендованої представниками Gradle. Він використовуватиме конвеєр маніпуляцій з байт-кодом Android, що є набагато більш логічним підходом. Основна ідея:

— змінити вихідні каталоги компіляції Kotlin і Java;

— скопіювати скомпільований вихідний код Kotlin/Java в один каталог;

— влаштувати АОП в комбіновані класи Kotlin/Java;

— зареєструвати генератор байт-коду, щоб Android розпізнавав користувацьке влаштування АОП як формальний крок у конвеєрі збірки.

Для цього необхідно створити PipelineAopWeaverPlugin, та унаслідувати його від Plugin<Project>, також додати константи, необхідні для побудови шляхів до файлів, що модифікуватимуться [17]:

```
private companion object {
```

```
        private const val MISSING_PLUGIN_ERROR =
"'com.android.application' or 'com.android.library' plugin required."
        private const val ANDROID_EXTENSION_NAME = "android"
        private const val ANDROID_JAR_TEMPLATE =
"%s/platforms/%s/android.jar"
        private const val PRE_WEAVE_DIR_TEMPLATE =
"preWeave/%s/%s"
        private const val POST_WEAVE_DIR_TEMPLATE =
"postWeave/%s"
        private const val PATTERN_ORIGINAL_KOTLINC_OUTPUT_DIR
= "tmp/kotlin-classes/"
        private const val AOP_WEAVE_TASK = "aopWeave%s"
        private const val AOP_LOG = "aop.log"
    }
```

Інтерфейс Plugin вимагає реалізації методу apply(), де відбувається весь основний код. Спочатку виконуються перевірки чи являється проект, що підключається додатком чи бібліотекою:

```
val isAndroid = project.plugins.hasPlugin(AppPlugin::class.java)
        val isLibrary = project.plugins.hasPlugin(LibraryPlugin::class.java)
        if (!isAndroid && !isLibrary) {
            throw GradleException(MISSING_PLUGIN_ERROR)
        }
        project.aopLog("Plugin started.")
        val extension =
project.extensions.create(AopWeaveExtension.AOP_WEAVE_EXTENSION,
AopWeaveExtension::class.java)
        val android =
project.extensions.findByName(ANDROID_EXTENSION_NAME) as
BaseExtension
        project.afterEvaluate {
```

```
val variants = if (isAndroid) {
    (android as AppExtension).applicationVariants
} else {
    (android as LibraryExtension).libraryVariants
}
```

Під час створення каталогів або пошуку завдань знадобляться деякі рядки в різних регістрових формах:

```
variants.forEach { variant ->
        val javaLangLowercase = LANG_JAVA.toLowerCase()
        val kotlinLangLowercase = LANG_KOTLIN.toLowerCase()
        val variantNameLowercase = variant.name
        val variantNameCapitalized = variant.name.capitalize()
```

Для виведення класів у каталоги, що використовуватимуться для коду, який не був модифікований за допомогою АОП, необхідно налаштувати завдання компіляції Kotlin і Java:

```
val preWeaveJavaDir = project.layout
        .buildDirectory
       .dir(PRE_WEAVE_DIR_TEMPLATE.format(variantNameLowercase,
javaLangLowercase))
    val preWeaveKotlinDir = project.layout
        .buildDirectory
       .dir(PRE_WEAVE_DIR_TEMPLATE.format(variantNameLowercase,
kotlinLangLowercase))
```

Саме тепер необхідно вивести класи Kotlin і Java модифіковані АОП:

```
val postWeaveDir = project.layout
        .buildDirectory
.dir(POST_WEAVE_DIR_TEMPLATE.format(variantNameLowercase))
```

Також отримуються постачальники завдань для кроків компіляції та власноруч зареєстроване завдання. Постачальники завдань дозволяють

повідомляти Gradle, як повинні бути налаштовані та виконані завдання, до того як вони створюються:

```
val kotlinCompileProvider =
project.kotlinCompileTaskProvider(variantNameCapitalized)
val javaCompileProvider =
project.javaCompileTaskProvider(variantNameCapitalized)
val kaptTaskProvider =
project.kaptTaskProvider(variantNameCapitalized)
val aopWeaveProvider = project.tasks
 .register(AOP_WEAVE_TASK.format(variantNameCapitalized),
AopWeaveTask::class.java)
val jacocoReportTaskProvider =
project.jacocoReportTaskProvider(variantNameCapitalized)
val extractAnnotationsTaskProvider =
project.extractAnnotationsTaskProvider(variantNameCapitalized)
```

Перш ніж Kapt (процесор анотацій Kotlin) зможе запуститися, потрібно відновити будь-які попередні класи з попередніх запусків завдань компіляції назад у цільові каталоги. Саме тут Kapt сподівається знайти ці класи. Якщо цього не зробити, інкрементні збірки не виконаються:

```
if (kaptTaskProvider != null) {
    configureKaptTask(kaptTaskProvider, kotlinCompileProvider,
javaCompileProvider, preWeaveJavaDir, preWeaveKotlinDir)
}
```

Також потрібно налаштувати завдання Kotlin/Java для виведення скомпільованих класів в інший каталог, ніж той, який він використовує за замовчуванням:

```
if (kotlinCompileProvider != null) {
    configureKotlinCompileTask(kotlinCompileProvider,
preWeaveKotlinDir)
}
```

```
if (javaCompileProvider != null) {
    configureJavaCompileTask(project, javaCompileProvider, preWeaveKotlinDir, preWeaveJavaDir)
}
```

Конфігурація для завдання JacocoReport на випадок, якщо цього вимагає збірка:

```
if (jacocoReportTaskProvider != null) {
    configureJacocoReportTask(jacocoReportTaskProvider, postWeaveDir)
}
```

Можна зіткнутися із завданням «extract<Variant>Annotations», яке очікує існування оригінального каталогу компіляції Kotlin. Потрібно переналаштувати його, щоб він працював правильно:

```
if (extractAnnotationsTaskProvider != null) {
configureExtractAnnotationsTask(extractAnnotationsTaskProvider, preWeaveKotlinDir)
}
```

Для влаштування АОП коду необхідний шлях до JAR файлу:

```
val androidJarPath = ANDROID_JAR_TEMPLATE.format(
    android.sdkDirectory.absolutePath,
    android.compileSdkVersion
)
```

На цьому моменті можна розпочинати конфігурацію АОП задачі для Gradle:

```
configureAopWeaveTask(
    project = project,
    extension = extension,
    variantNameCapitalized = variantNameCapitalized,
    javaCompileProvider = javaCompileProvider,
    kotlinCompileProvider = kotlinCompileProvider,
```

```
                aopWeaveProvider = aopWeaveProvider,
                androidJarPath = androidJarPath,
                preWeaveJavaDir = preWeaveJavaDir,
                preWeaveKotlinDir = preWeaveKotlinDir,
                postWeaveDir = postWeaveDir
            )
```

Останній крок – дати зрозуміти логіці збірки Android, що вона повинна очікувати модифікованих байт-кодом класів. Це механізм, який дозволяє додати власну роботу в конвеєр перед остаточним складанням модуля:

```
variant.registerPostJavacGeneratedBytecode(project.files(aopWeaveProvide
r.map { it.outputDir!! }))
```

Функція configureKaptTask() переміщує будь-які класи попередньої компіляції, згенеровані попередніми завданнями, назад у цільові каталоги. Після чого розпочинає запуск Kapt та копіює класи назад із цільових каталогів у каталоги попередньої компіляції. Це забезпечує належну роботу поетапних збірок:

```
private fun configureKaptTask(
        kaptTaskProvider: TaskProvider<Task>,
        kotlinCompileTaskProvider: TaskProvider<Task>?,
        javaCompileTaskProvider: TaskProvider<Task>?,
        preWeaveKotlinDir: Provider<Directory>,
        preWeaveJavaDir: Provider<Directory>
    ) {
        kaptTaskProvider.configure {
            swapPreWeaveContent(kotlinCompileTaskProvider,
preWeaveKotlinDir)
            swapPreWeaveContent(javaCompileTaskProvider,
preWeaveJavaDir)
        }
    }
```

Функція configureKotlinCompileTask() налаштовує компілятор Kotlin для виведення в каталог попередньої компіляції. Також вона підключає деякі дії, які будуть переміщувати скомпільовані класи. Це зроблено для збереження функціональних можливостей інкрементної збірки та підготовки до влаштування АОП:

```
private fun configureKotlinCompileTask(
    kotlinCompileProvider: TaskProvider<Task>,
    preWeaveKotlinDir: Provider<Directory>
) {
    kotlinCompileProvider.configure {
        outputs.dir(preWeaveKotlinDir)
        swapPreWeaveContent(this, preWeaveKotlinDir)
    }
}
```

Аналогічним чином працює функція configureJavaCompileTask(), виконуючи вищеперечислені дії з Java компілятором.

У певний момент змін у версіях 7.0.X плагіна Android Gradle, "extract<Variant>Annotations" почав з'являтися відразу після компіляції Kotlin. Це завдання почало давати збій, оскільки воно більше не могло знайти каталог «extract<Variant>Annotations».

Оскільки цей каталог ніколи не створювався через те, як керуються вихідні каталоги в задачах компіляції (щоб дозволити здійснювати влаштування АОП), потрібно переконатися, що відбуваються дві речі.

По-перше, чи завдання "extract<Variant>Annotations" не завершилося через відсутність каталогу. Для цього достатньо переконатись, що цей каталог існує, як обхідний шлях.

По-друге, чи фактичний вихідний каталог компіляції Kotlin включений у список, необхідний для цього завдання. "classpath" — це той шлях, що було б зручно змінити, але він незмінний. Однак, дивлячись на логіку в класі

"ExtractAnnotations", він копіює все від "bootClasspath" до "classpath" перед виконанням, а "bootClasspath" змінюється. Отже, якщо додати справжній вихідний каталог компіляції до "bootClasspath", він підбере скомпільовані класи правильно і працюватиме належним чином. Саме це відбувається у реалізації функції configureExtractAnnotationsTask:

```
private fun configureExtractAnnotationsTask(
    extractAnnotationsProvider: TaskProvider<ExtractAnnotations>,
    preWeaveKotlinDir: Provider<Directory>
) {
    extractAnnotationsProvider.configure {
        doFirst {
            classpath.files.forEach {
                if
(it.path.contains(PATTERN_ORIGINAL_KOTLINC_OUTPUT_DIR) &&
!it.exists()) {
                    it.mkdirs()
                }
            }
        }
        bootClasspath += project.files(preWeaveKotlinDir)
    }
}
```

3.2 Використання АОП для збору аналітичних даних

АОП дозволяє різні варіанти індикації коду, що повинен бути модифікований. Найбільш доцільним для збору аналітики є позначення функцій анотаціями, оскільки необхідно керувати тим, які методи повинні бути відслідковані, що легко досягається позначенням методу відповідною анотацією. Необхідно створити дві анотації: одна відповідатиме за відправку даних до спрацювання коду метода, інша ж – після. Це є корисним у випадку

коли метод певним чином впливає на зміну даних, що повинні відслідковуватись. Позначення, що анотація повинна бути застосована лише для методів:

```
@Target(AnnotationTarget.FUNCTION)
annotation class TrackBefore()
@Target(AnnotationTarget.FUNCTION)
annotation class TrackAfter()
```

Код роботи з АОП розташовано в класі AnalyticsAspect.kt та позначено анотацією @Aspect, що дозволяє його використання на етапі компіляції вихідного коду. PointCut — це набір з однієї або кількох точок приєднання, де має виконуватися відповідний код. Необхідно додати точки приєднання для створених анотацій:

```
@Pointcut("@annotation(TrackBefore)")
   fun trackBeforeAspect() {
   }
   @Pointcut("@annotation(TrackAfter)")
   fun trackAfterAspect() {
   }
```

Для відслідковування моментів до і після виконання методів, використовуються відповідні анотації @Before та @After:

```
@Before("@annotation(TrackBefore) && execution(@TrackBefore *
*.*(..))")
   fun before(joinPoint: JoinPoint) {
      Log.d("Aspect", "Before is called")
      val event = createEvent(joinPoint)
      EventsTracker.track(event)
   }
   @After("@annotation(TrackAfter) && execution(@TrackAfter *
*.*(..))")
   fun after(joinPoint: JoinPoint) {
```

```
        Log.d("Aspect", "After is called")
        val event = createEvent(joinPoint)
        EventsTracker.track(event)
    }
```

Код, що знаходиться всередині цих методів створює подію та відправляє її до класу, що відповідає безпосередньо за їх обробку. Процес формування події:

```
    private fun createEvent(joinPoint: JoinPoint): Event {
        val methodName = joinPoint.signature.name
        val parameterNames = (joinPoint.signature as
CodeSignature).parameterNames
        val parameterValues = joinPoint.args
        val namesToValues = parameterNames.zip(parameterValues).toMap()
        return Event(methodName, namesToValues)
    }
```

Передавши об’єкт точки приєднання можна отримати можливість діставання метаданих сигнатури метода. Таким чином, доцільно використати ім’я методу як назву для події. Назви ж параметрів методу слугуватимуть ключами для аналітичних даних і вони, разом із самими ж даними, поміщаються в словник та передаються в конструктор класу події.

3.3 Ініціалізація бібліотеки

Розглянемо головний клас Analytics.kt, що використовується для ініціалізації бібліотеки. Оскільки використано шаблон Builder, конструктор самого класу позначений приватним для забезпечення його створення лише через метод build():

```
    private constructor(
        activity: Activity,
        navController: NavController?,
        screensToOmit: Set<Int>
```

)

Context – це один із найважливіших компонентів системи Android, який відповідає за роботу з ресурсами додатку, стилями, створенням відображень (View), запуском сторонніх активностей та багато чого іншого. Даний клас є абстрактним і його реалізовують основні компоненти системи Android, такі як додаток (Application), фрагмент (Fragment) та активність (Activity) [17]. Використання фрагментів та активностей у сінглтонах призводить до витоків пам’яті, чого допускати не можна, саме тому при збережені контексту до змінної є можливість отримати контекст додатку, що є безпечним, оскільки сінглтон та додаток мають однаковий час життя:

private val context: Context = activity.applicationContext

Для перевірки статусу згоди користувача на обробку аналітичних даних, створюється відповідний клас помічник, який розглянутий детальніше у підрозділі, присвяченому обробці згоди користувача:

private val consentPreferencesHelper: ConsentPreferencesHelper =

ConsentPreferencesHelper(context)

За стан бібліотеки відповідає відповідна змінна state, яка при зміні значення виконує перевірку на активність стану. У випадку, якщо стан активний — відсилаються накопичені за час ініціалізації події, а також запускається періодична задача для відправки подій через певні проміжки часу. Якщо ж стан переходить у вимкнений, відбувається очистка подій і завершення періодичної задачі за її наявності:

```
var state: State = State.PendingInitialization
    private set(value) {
        field = value
        if (value is State.Enabled) {
            EventsTracker.trackPendingEvents()
            EventsTracker.startPeriodicalJob(value.configuration.timerSeconds)
        } else {
```

```
            EventsTracker.removePendingEvents()
            EventsTracker.cancelPeriodicalJob()
        }
    }
```

Для отримання конфігурації використано гетер, що полегшує доступ, оминаючи виклик змінної стану:

```
val configuration: RemoteConfiguration?
        get() = state.configuration
```

У випадку зміни користувачем згоди на обробку даних необхідно змінити поточний стан, для цього додано відповідний метод, що перемикає стан:

```
fun onConsentChanged(isConsentGiven: Boolean) {
        state = State.pickState(isConsentGiven, state.configuration)
    }
```

Реалізований клас Builder, приймає у якості параметрів активність для отримання контексту, NavController для реалізації відслідковування екранів та список ідентифікаторів екранів, які по певних причинах повинні бути виключені з переліку відслідковуваних:

```
class Builder {
    private var navController: NavController? = null
    private var screensToOmit: Set<Int> = emptySet()
    fun setNavController(navController: NavController): Builder {
        this.navController = navController
        return this
    }
    fun setScreensToOmit(ids: Set<Int>): Builder {
        this.screensToOmit = ids
        return this
    }
    fun build(activity: Activity): Analytics {
```

```
            return Analytics(activity, navController, screensToOmit)
        }
    }
```

Для викачування конфігураційного файлу додано об’єкт репозиторія і CoroutineScope для забезпечення асинхронності викликів:

```
private val coroutineScope = CoroutineScope(Dispatchers.IO)
private val configRepository = ConfigurationRepository()
```

Оскільки контекст необхідний в багатьох файлах бібліотеки, необхідно забезпечити вільний доступ до нього. Тому в блоці ініціалізації передано контекст у відповідний сінглтон, що відповідатиме за його надання іншим класам та об’єктам:

```
ContextProvider.onInitialization(context.applicationContext)
```

В асинхронному блоці проведено базові дії щодо ініціалізації. Спершу завантажено події, які були записані до пам’яті пристрою, але не були відправлені у зв’язку з передчасним завершенням роботи додатку. Для роботи бібліотеки необхідний файл конфігурації, який завантажується та на його основі визначається стан, в який необхідно перевести бібліотеку. Наостанок зареєстровано лістенер на зміну поточного екрану. Описаний код ініціалізації:

```
coroutineScope.launch {
            EventsTracker.loadExistingEvents()
            val remoteConfig = configRepository.fetchConfiguration()
            val configuration = remoteConfig.mapToConfiguration()
            val consentGiven = consentPreferencesHelper.isConsentGiven() ==
true
            state = State.pickState(consentGiven, configuration)
        }
navController?.addOnDestinationChangedListener(NavigationDestinationLi
stener(screensToOmit))
```

Стан бібліотеки обробляється за допомогою ізольованих класів. Даний тип класів використовується у мові Kotlin як заміна традиційному enum

завдяки можливості включати в деякі стани різні дані. Таким чином у станах Enabled та Disabled зберігається конфігурація, у випадку ж зі станом NotInitialized конфігурація відсутня, оскільки вона ще не була викачана. Лістинг класу State:

```
sealed class State(open val configuration: RemoteConfiguration?) {
    data class Enabled(override val configuration: RemoteConfiguration):
State(configuration)
    data class Disabled(override val configuration: RemoteConfiguration):
State(configuration)
    object PendingInitialization: State(null)
}
```

Для правильного перемикання станів додано функцію, що визначає стан на основі параметрів, що передаються:

```
companion object {
        fun pickState(isConsentGiven: Boolean, configuration:
RemoteConfiguration?): State {
            return when {
                configuration == null -> PendingInitialization
                isConsentGiven && configuration.enabled ->
Enabled(configuration)
                else -> Disabled(configuration)
            }
        }
    }
```

3.4 Реалізація відслідковування зміни екранів

Для обробки зміни екранів у блоці ініціалізації бібліотеки було зареєстровано NavigationDestinationListener. Даний клас наслідує NavController.OnDestinationChangedListener, що є інтерфейсом, реалізацію

якого очікує отримати NavController. В конструктор класу передано набір ідентифікаторів екранів, які повинні бути проігноровані при відслідковуванні:

```
class NavigationDestinationListener(private val screensToOmit: Set<Int>) :
    NavController.OnDestinationChangedListener
```

В класі реалізується єдиний метод onDestinationChanged:

```
override fun onDestinationChanged(
        controller: NavController,
        destination: NavDestination,
        arguments: Bundle?
    ) {
        val destinationId = destination.id
        if (!screensToOmit.contains(destinationId)) {
            EventsTracker.changeScreen(destination.label as String)
        }
    }
```

За допомогою параметра destination отримується ідентифікатор екрану та перевіряється на вміст у screensToOmit. Якщо ідентифікатор відсутній у колекції – відслідковування екрану дозволено. Екрани відслідковуються викликом відповідного методу changeScreen класу EventsTracker, що приймає назву екрану.

У випадку зі змінами сторінок ViewPager постає необхідність у створенні власної реалізації шляхом наслідування. Створений клас AnalyticsViewPager:

```
class AnalyticsViewPager(context: Context, attrs: AttributeSet? = null) :
ViewPager(context, attrs)
```

Додано лістенер на зміну екрану:

```
private val listener = object : OnPageChangeListener {
        override fun onPageSelected(position: Int) {
```

Об’єкт адаптера приведено до AnalyticsViewPagerAdapter, а фрагмент отримується за допомогою відповідного методу:

```
            (adapter as AnalyticsViewPagerAdapter?)?.let {
                val fragment = it.getFragment(position)
```

У випадку якщо фрагмент вже знаходиться у стані resumed – відслідковується екран, інакше – за допомогою методу doOnResume() відслідковування станеться коли фрагмент перейде у даний стан:

```
                    if (fragment?.isResumed == true) {
                    trackScreenChange(position)
                } else {
                    fragment?.doOnResume { trackScreenChange(position) }
                }
            }
        }
```

Оскільки інші методи лістенера не потребуються, для них задано пусту реалізацію через Unit:

```
        override fun onPageScrolled(
            position: Int,
            positionOffset: Float,
            positionOffsetPixels: Int
        ) = Unit
        override fun onPageScrollStateChanged(state: Int) = Unit
```

У методі trackScreenChange() в залежності від позиції фрагмента отримується назва сторінки, яка потім передається у відповідний метод changeScreen():

```
        private fun trackScreenChange(position: Int) {
            val title = adapter?.getPageTitle(position)?.toString().orEmpty()
            EventsTracker.changeScreen(title)
        }
    }
```

В блоці ініціалізації підписується лістенер:

```
init {
```

```
        addOnPageChangeListener(listener)
    }
```

Оскільки створений ViewPager повинен працювати лише з адаптерами типу AnalyticsViewPagerAdapter у перевизначеному методі setAdapter() об’єкт адаптера перевіряється на відповідність і у випадку, якщо результат негативний — викидається помилка:

```
override fun setAdapter(adapter: PagerAdapter?) {
        if (adapter !is AnalyticsViewPagerAdapter) {
            throw IllegalArgumentException("Provided adapter not supported. Dedicated adapter required")
        }
        super.setAdapter(adapter)
    }
```

Метод doOnResume реалізується як функція розширення до класу фрагмента та приймає блок коду, що має виконатись фрагментом при входженні в стан resumed:

```
fun Fragment.doOnResume(block: () -> Unit) {
        val lifecycleObserver = object : LifecycleObserver {
```

Метод, що має бути виконаний, позначається відповідною анотацією. У тілі метода перед виконанням блоку коду відбувається відписка lifecycleObserver для того, щоб блок коду спрацював лише однин раз:

```
            @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
            fun runBlock() {
                this@doOnResume.lifecycle.removeObserver(this)
                block()
            }
        }
```

LifecycleObserver підписується до життєвого циклу фрагмента:

```
        lifecycle.addObserver(lifecycleObserver)
```

```
    }
```

Необхідно створити абстрактний клас адаптера для наслідування кодом клієнтського додатку:

```
abstract class AnalyticsViewPagerAdapter : PagerAdapter() {
```

Словник зберігає додані фрагменти разом із позицією:

```
    private val registeredFragments: MutableMap<Int, Fragment> = HashMap()
```

При створенні фрагменту він заноситься до словника під відповідною позицією:

```
    override fun instantiateItem(container: ViewGroup, position: Int): Any {
        val fragment = super.instantiateItem(container, position)
        registeredFragments[position] = fragment as Fragment
        return fragment
    }
```

Щоб отримувати фрагмент за позицією, необхідно створити відповідний метод:

```
    fun getFragment(position: Int): Fragment? {
        return registeredFragments[position]
    }
}
```

3.5 Реалізація обробки аналітичних подій

Функція track() є обробником усіх аналітичних подій:

```
fun track(event: Event) {
        when (Analytics.state) {
```

У випадку коли бібліотека у ввімкненому стані — будується розширена подія, а потім або відсилається на сервер, або зберігається до пам’яті, залежно від того, чи була виконана умова по кількості накопичених подій:

```
            is State.Enabled -> {
```

```
                val extendedEvent = buildExtendedEvent(event)
                events.add(extendedEvent)
                coroutineScope.launch {
                    if (meetsSendingConditions()) {
                        trackEvents(events)
                    } else {
                        FileIO.saveToFile(events, EVENTS_FILE_NAME)
                    }
                }
            }
```

Коли конфігурація ще не була отримана, події накопичуються до моменту її завантаження:

```
                State.PendingInitialization -> {
                    pendingEvents.add(event)
                }
                is State.Disabled -> {}
            }
        }
```

Функція meetsSendingConditions() перевіряє чи кількість накопичених подій збігається з отриманою через конфігурацію:

```
    private fun meetsSendingConditions(): Boolean {
        return config?.eventsPackageSize?.let { events.size >= it } ?: false
    }
```

Функція changeScreen() зберігає назву екрану в змінну для подальшого використання при побудові наступних подій, а також генерує відповідну подію та відправляє через метод track():

```
    fun changeScreen(screenName: String) {
        this.screen = screenName
        val screenEvent = buildScreenNameEvent()
        track(screenEvent)
```

```
    }
```

При побудові розширеної події з об’єкту конфігурації дістається набір параметрів для отриманої події та разом із сховищем параметрів та назвою екрана передається у клас EventBuilder, що відповідає за перетворення об’єкта:

```
private fun buildExtendedEvent(event: Event): Event {
    val scope = config?.events?.get(event.name) ?: emptySet()
    return EventBuilder(event)
        .setEventScope(scope)
        .setStorageParams(StatefulStorage.parameters)
        .setScreen(screen)
        .build()
}
```

Аналогічним чином працює метод buildScreenNameEvent для створення події про зміну екрану, але в даному випадку подія створюється з нуля.

Оскільки за час, який займала ініціалізація могли накопичитись події, створюємо відповідний метод, що відправлятиме їх при ввімкненні бібліотеки:

```
fun trackPendingEvents() {
    coroutineScope.launch {
        trackEvents(pendingEvents)
    }
}
```

У випадку ж деактивації, накопичені події повинні бути видалені, щоб не займати вільну пам’ять:

```
fun removePendingEvents() {
    pendingEvents.clear()
}
```

Періодична робота, необхідна для відправки подій через задані проміжки часу, реалізується за допомогою функції delay() у нескінченному циклі. Робота зберігається у змінну класу для можливості її зупинки у майбутньому:

```
fun startPeriodicalJob(periodSeconds: Long) {
    periodicalJob?.cancel()
    periodicalJob = coroutineScope.launch {
      val millis = TimeUnit.SECONDS.toMillis(periodSeconds)
      while (true) {
        delay(millis)
        trackEvents(events)
      }
    }
  }
```

Для зупинки роботи реалізовано окремий метод, що виконується за її наявності:

```
fun cancelPeriodicalJob() {
    periodicalJob?.cancel()
  }
```

Метод loadExistingEvents() завантажує події, що не були відправленні в зв’язку з непередбаченим завершенням додатку:

```
suspend fun loadExistingEvents() {
    val existingEvents =
FileIO.readFromFile<MutableList<Event>>(EVENTS_FILE_NAME) ?: return
    events.addAll(existingEvents)
  }
```

Оскільки завантаження може відбуватися з різних місць і повинно виконуватись в фоновому потоці, виникає потреба в синхронізації виконання. У випадку з корутинами традиційні методи синхронізації на зразок Synchronized не працюватимуть, оскільки корутина, запущена на одному потоці, може завершити своє виконання на іншому. Для вирішення проблеми синхронізації існує спеціальний клас Mutex. Синхронізація виконується за допомогою об’єкта, що розташований в об’єкті:

```
private suspend fun trackEvents(events: List<Event>) {
```

```
            mutex.withLock {
                eventsRepository.trackEvents(events)
                this.events.clear()
                FileIO.saveToFile(events, EVENTS_FILE_NAME)
            }
        }
```

Окрім відправки даних на сервер, відбувається очистка даних та перезапис до пам’яті пристрою.

Для модифікації базових подій використовується клас EventBuilder. Відповідно до назви даний клас використовує шаблон проектування Builder. Як обов’язкові параметри приймаються назва події та словник з параметрами:

```
class EventBuilder(private var name: String, private var eventParameters: Map<String, Any>)
```

Для розширення події було б зручніше працювати з конструктором, який приймає клас існуючої події. Тому має сенс перевантажити конструктор:

```
constructor(event: Event): this(event.name, event.params)
```

Для модифікації події використовуються такі параметри як додаткові параметри зі сховища, план події для фільтрації та назва поточного екрану. Для задання всіх трьох параметрів додано відповідні методи:

```
fun setStorageParams(params: Map<String, Any>): EventBuilder {
    this.storageParameters = params
    return this
}
fun setEventScope(scope: Set<String>): EventBuilder {
    this.scope = scope
    return this
}
fun setScreen(screen: String): EventBuilder {
    this.screen = screen
    return this
```

```
    }
```

Метод build() виконує операції модифікування. Спершу відбувається додавання параметрів зі сховища, при цьому дані з події мають пріоритет у випадку збігу ключів. За набором ключів для параметрів відбувається фільтрування та додається параметр з іменем екрану:

```
fun build(): Event {
    val params: MutableMap<String, Any> = HashMap(storageParameters)
    params.putAll(eventParameters)
    params.addScreenNameParameter()
    val scopedKeys = params.keys.intersect(scope)
    val scopedParameters = params.filter { scopedKeys.contains(it.key) }
    return Event(name, scopedParameters)
}
```

Параметр імені екрану додається з використанням константи, що знаходиться у класі RemoteConfiguration:

```
private fun MutableMap<String, Any>.addScreenNameParameter() {
    if (screen.isNotEmpty()) {
        put(RemoteConfiguration.SCREEN_EVENT_NAME, screen)
    }
}
```

Робота з файлами реалізована за допомогою механізму Java ObjectInputStream та ObjectOutputStream. Інтернет виклики виконуються з використанням бібліотеки Retrofit 2, оскільки дана бібліотека підтримує нативну роботу з корутинами та надає можливості автоматичного парсингу даних завдяки GsonConverterFactory.

Сховище параметрів StatefulStorage реалізовано у вигляді сінглтона з базовими методами для додавання параметра, його видалення та очищення всіх існуючих параметрів:

```
internal val parameters: MutableMap<String, Any> = HashMap()
    fun addParameter(key: String, value: Any) {
```

```
        parameters[key] = value
    }
    fun removeParameter(key: String) {
        parameters.remove(key)
    }
    fun clear() {
        parameters.clear()
    }
```

3.6 Реалізація запиту на обробку особистих даних

Для перевірки стану згоди необхідний клас, який інкапсулює в собі роботу зі зчитуванням та записом рішення користувача. Один метод отримує дані та виконує запис, інший — повертає стан за його наявності або null, якщо запит досі не відбувався. Стан зберігається за допомогою SharedPreferences системи Android, що забезпечує швидкий доступ синхронізацію:

```
    companion object {
        private const val CONSENT_GIVEN_KEY = "pref_consent_given"
    }
    private val sharedPreferences =
        context.getSharedPreferences("ContestPreferences",
Context.MODE_PRIVATE)
    fun isConsentGiven(): Boolean? {
        return if (sharedPreferences.contains(CONSENT_GIVEN_KEY)) {
            sharedPreferences.getBoolean(CONSENT_GIVEN_KEY, false)
        } else {
            null
        }
    }
    fun setConsent(value: Boolean) {
```

```
        return sharedPreferences.edit().putBoolean(CONSENT_GIVEN_KEY,
value).apply()
    }
```

Для відображення запиту необхідно використати окреме Activity, оскільки екран повинен відображатись в момент, зручний для клієнтського додатку. Окрема активність дозволяє відображати екран поверх існуючого, таким чином не втручаючись до навігації. Ініціалізацію достатньою виконати у методі onCreate() життєвого циклу:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_consent)
    consentTitle = findViewById(R.id.title)
    consentSubtitle = findViewById(R.id.subtitle)
    agreeButton = findViewById(R.id.agree_button)
    declineButton = findViewById(R.id.decline_button)
```

За наявності конфігурації екрану необхідно оновити дані:

```
    val configuration = Analytics.configuration?.consentScreenData
    if (configuration != null) {
       setupConsentScreen(configuration)
    }
```

Відповідні лістенери змінюватимуть стан згоди:

```
    agreeButton.setOnClickListener {
       updateConsent(true)
    }
    declineButton.setOnClickListener {
       updateConsent(false)
    }
  }
```

Дані екрану оновлюються за наявністю відповідних полів у конфігурації. Якщо певне поле відсутнє, використовується значення за замовчуванням:

```
private fun setupConsentScreen(data: ConsentScreenData) {
    data.title?.let { consentTitle.text = it }
    data.subtitle?.let { consentSubtitle.text = it }
    data.agreeButtonText?.let { agreeButton.text = it }
    data.declineButtonText?.let { declineButton.text = it }
}
```

Метод оновлення згоди зберігає значення до пам’яті, перемикає стан бібліотеки та завершує відкриту активність:

```
private fun updateConsent(value: Boolean) {
    preferencesHelper.setConsent(value)
    Analytics.onConsentChanged(value)
    finish()
}
```

UML діаграма створеної програми наведена в Додатку Е.

4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ДОДАТКУ

Для тестування додатку створено простий шаблон з декількома екранами, що здатний продемонструвати базові функціональні можливості.

Щоб виконати підключення додатку необхідно додати відповідні залежності у конфігураційний файл збірника, наприклад Gradle, Maven або Ant. Команда підключення для Gradle виглядає наступним чином:

implementation 'com.entermannn.mastersproject'

Оскільки додаток не опублікований в системі Maven Central, синхронізація залежностей не відбудеться. Для коректної роботи потрібно підключити додаток локально. Найбільш легкий спосіб цього досягнути – скористатися системою залежностей Maven Local, яка базується на локальному комп’ютері. Для цього необхідно виконати наступні команди: clean, assemble та publishToMavenLocal. Дані команди вказують збірнику очистити закешовані значення з попередніх збірок, після чого побудувати нову версію та опублікувати її в системі Maven Local.

У тестовому додатку необхідно додати репозиторій Maven Local:

```
repositories {
        mavenLocal()
    }
```

Так як додаток цілком покладається на віддалену конфігурацію потрібно її створити для забезпечення працездатності:

```
{
    "enabled": "true",
    "packageSize": 15,
    "timerSeconds": 1,
    "events": {
        "Screen changed": ["screen"],
        "recommend": ["author", "genre", "title", "year", "review", "score",
"screen"],
```

```
        "notRecommend": ["genre", "title", "year", "review", "screen"]
    }
}
```

Параметр «timerSeconds» приймає значення 1, щоб події відправлялися кожної секунди, що дозволить значно пришвидшити перевірку. Початковий екран тестового додатку зображено на рисунку 4.1.

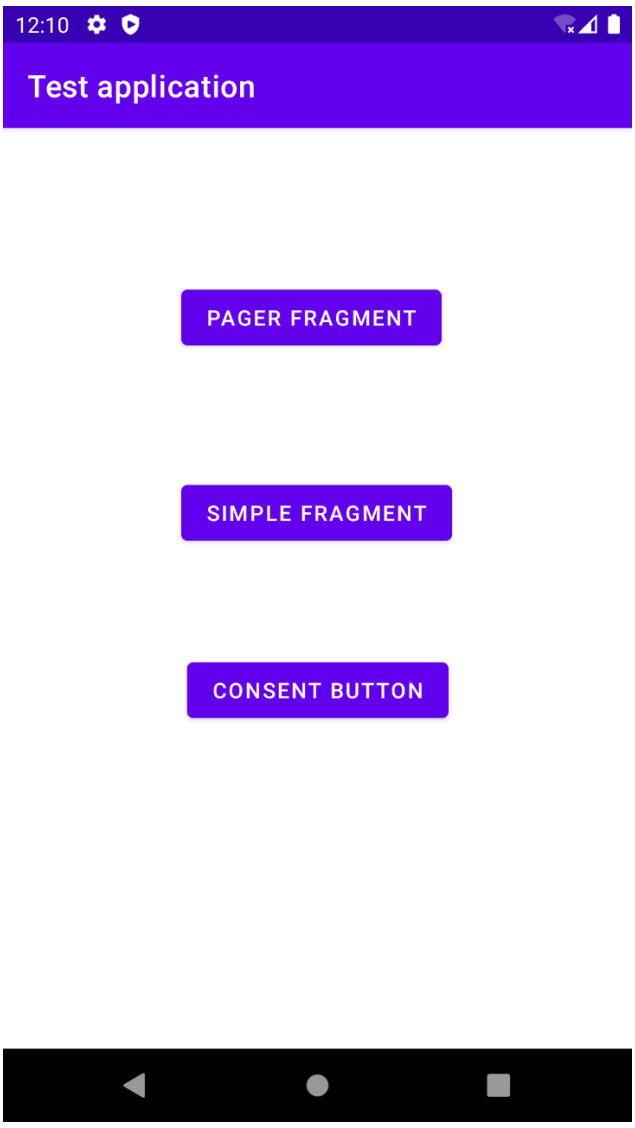

Рисунок 4.1 – Початковий екран тестового додатку

Для подій «recommend» та «notRecommend» вказані різні набори параметрів. Це було зроблено з метою продемонструвати роботу процесу фільтрації параметрів залежно від конфігурації.

В тестовому додатку необхідно провести ініціалізацію бібліотеки. Даний процес виглядає таким чином:

```
val navHostFragment =
supportFragmentManager.findFragmentById(R.id.nav_host_fragment_activity_main) as NavHostFragment
        navController = navHostFragment.navController
        analytics = Analytics.Builder(this)
           .setNavController(navController)
           .setScreensToOmit(setOf(R.id.pagerFragment))
           .build()
```

Для відображення подій додано логування за допомогою команди Android Log.d():

```
Log.d("TestApplication", events.toString())
```

Дана команда виводитиме в консоль назву події та словник із ключами та параметрами.

При запуску додатку та переході між екранами можна помітити, що події не відправляються. Це цілком очікувана поведінка, оскільки для ввімкнення відслідковування також необхідна згода користувача на обробку персональних даних. Для цього необхідно перейти на екран «Consent» через «Consent button» (рисунок 4.2):

Після натиснення кнопки «Agree» події починають відправлятися в штатному режимі:

```
2021-12-14 12:37:37.767 27779-27808/com.entermannn.mastersproject D/TestApplication: [Event(name=Screen changed, params={screen=Simple Fragment})]
2021-12-14 12:38:21.883 27779-27808/com.entermannn.mastersproject D/TestApplication: [Event(name=Screen changed, params={screen=Start Fragment})]
```

Simple Fragment представляє собою шаблон відображення даних про книгу з можливістю залишення відгуку, оцінки та рекомендації книги.

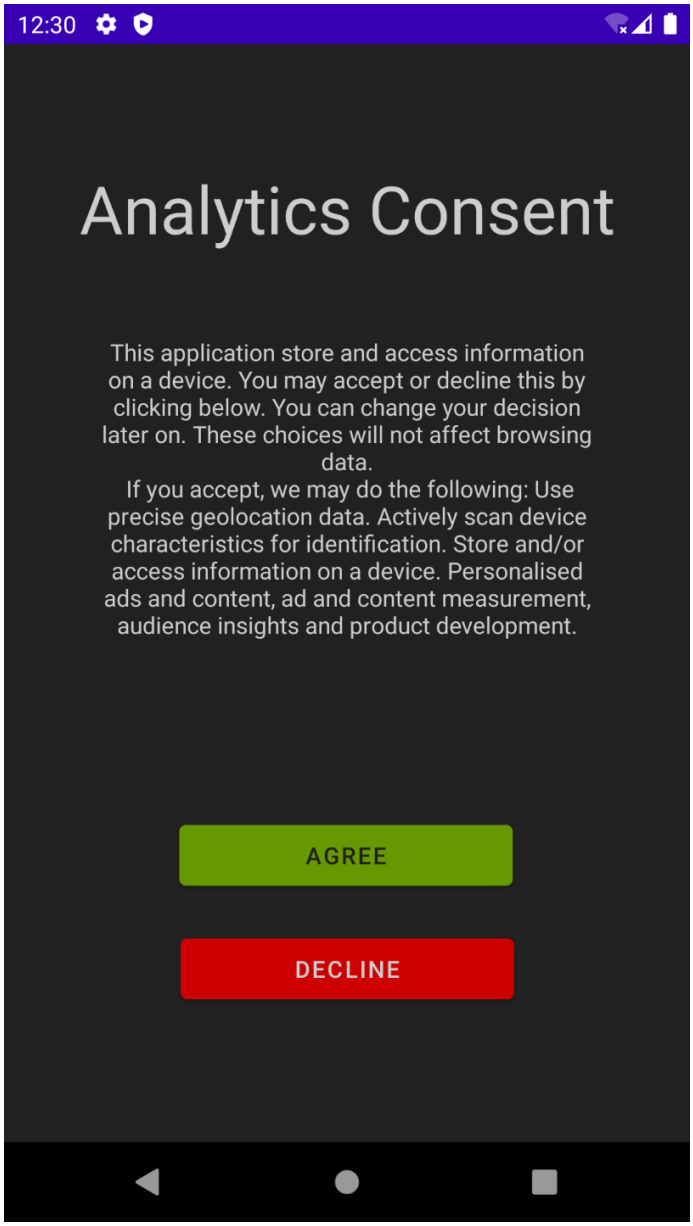

Рисунок 4.2 – Екран згоди користувача на обробку особистих даних

Структура фрагменту складається з набору текстових полів та двух кнопок з відповідними обробниками (рисунок 4.3):

review = view.findViewById(R.id.review)

score = view.findViewById(R.id.score)

recommendButton = view.findViewById(R.id.recommend)

notRecommendButton = view.findViewById(R.id.not_recommend)

title = view.findViewById(R.id.title)

year = view.findViewById(R.id.year)

author = view.findViewById(R.id.author)

```
genre = view.findViewById(R.id.genre)
recommendButton.setOnClickListener {
    recommend(review.text.toString(), score.text.toString())
}
notRecommendButton.setOnClickListener {
    notRecommend(review.text.toString(), score.text.toString())
}
```

Два методи, що обробляють дії з рекомендацією позначено відповідними анотаціями для відслідковування:

```
@TrackBefore
private fun recommend(review: String, score: String) {
    /// Recommending
}
@TrackBefore
private fun notRecommend(review: String, score: String) {
    /// Not recommending
}
```

Параметри, що відносяться до контексту екрану задаються через StatefulStorage:

```
StatefulStorage
        .addParameter("title", title.text.toString())
        .addParameter("year", year.text.toString())
        .addParameter("author", author.text.toString())
        .addParameter("genre", genre.text.toString())
```

Дані параметри повинні включатись до будь-яких подій, відправлених з цього екрану.

При натисненні кнопок Recommend та Not recommend було відправлено дві події з відповідними назвами та набором параметрів відповідно до конфігурації:

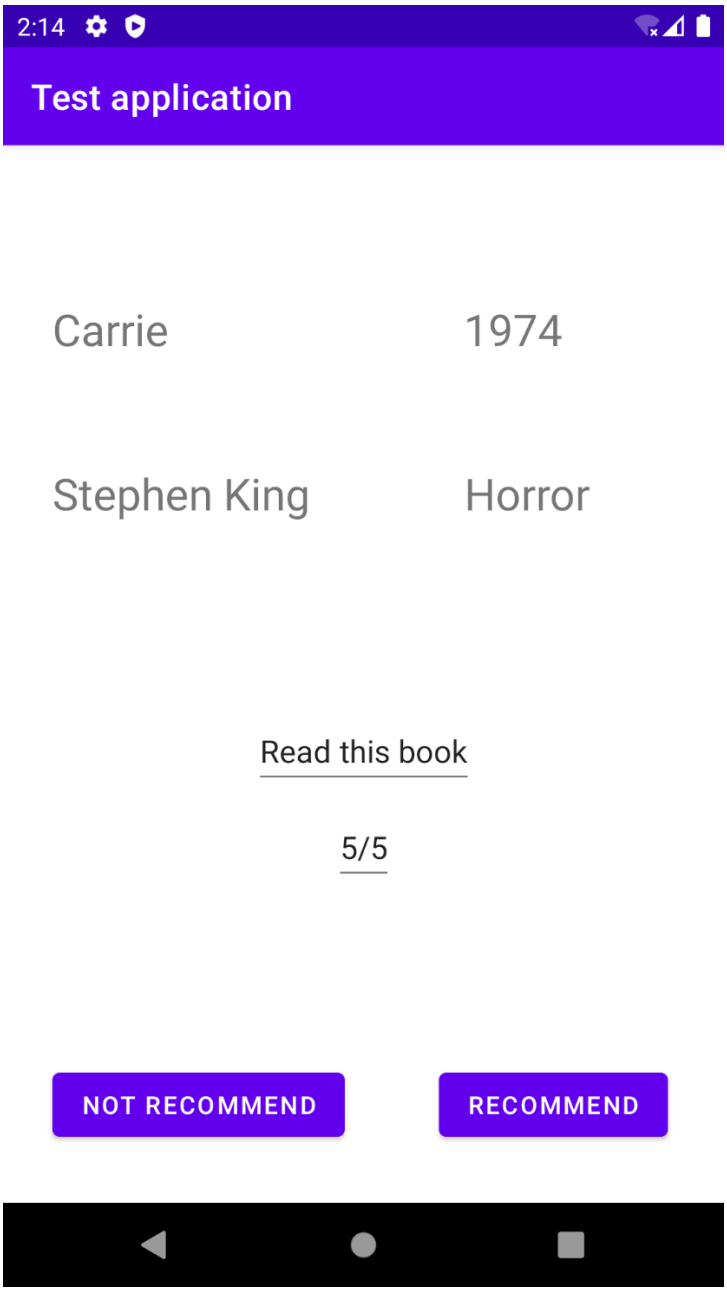

Рисунок 4.3 – Відображення Simple Fragment

2021-12-14 14:25:21.034 27779-27807/com.entermannn.mastersproject D/TestApplication: [Event(name=recommend, params={score=5/5, year=1974, author=Stephen King, review=Read this book, genre=Horror, screen=Simple Fragment, title=Carrie})]

2021-12-14 14:31:49.003 27779-27807/com.entermannn.mastersproject D/TestApplication: [Event(name=notRecommend, params={year=1974, review=Read this book, genre=Horror, screen=Simple Fragment, title=Carrie})]

Pager Fragment представляє собою екран з відображеним на ньому ViewPager, що дозволяє перелистувати сторінки (рисунок 4.4).

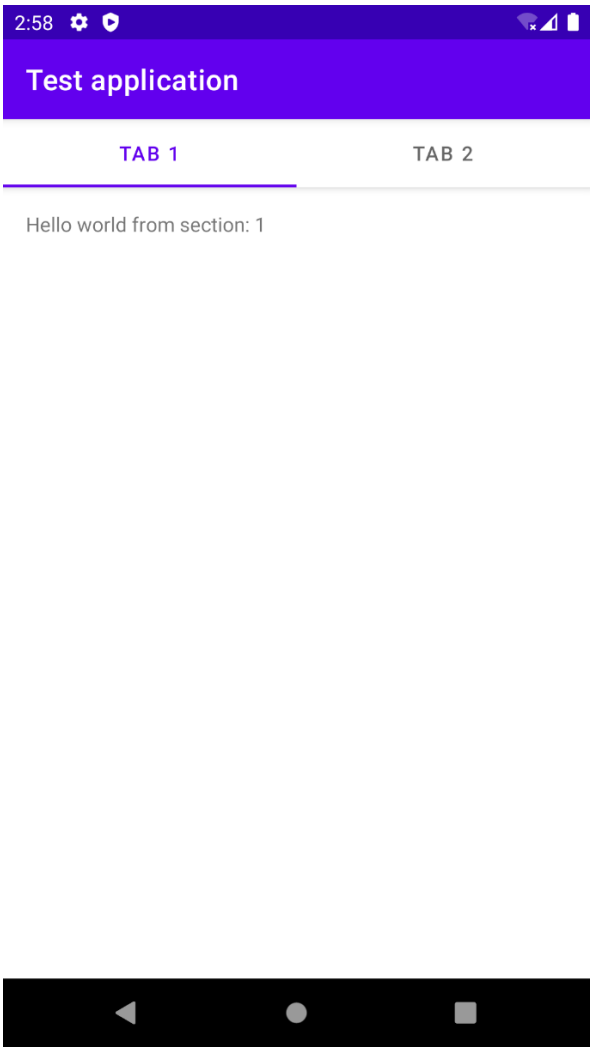

Рисунок 4.4 – Відображення Pager Fragment

Як компонент ViewPager використано AnalyticsViewPager разом з AnalyticsViewPagerAdapter, що дозволяє відслідковувати зміну екранів на метод onResume() життєвого циклу. Для перевірки перевизначено метод onStart() фрагменту, що відповідає за конкретні сторінки. Таким чином, буде вказано, що фрагмент увійшов в активний стан:

```
override fun onStart() {
    super.onStart()
    Log.d("TestApplication", "Pager fragment is active")
}
```

При переході на Pager Fragment, або при переключенні сторінок спочатку повинен спрацьовувати метод onStart() child фрагменту, а потім відслідковуватись екран. Це можна побачити переглянувши логування:

2021-12-14 15:07:12.176 31740-31740/com.entermannn.mastersproject D/TestApplication: Pager fragment is active

2021-12-14 15:07:12.178 31740-31740/com.entermannn.mastersproject D/TestApplication: Pager fragment is active

2021-12-14 15:07:12.619 31740-31771/com.entermannn.mastersproject D/TestApplication: [Event(name=Screen changed, params={screen=Tab 1})]

В даному випадку два фрагменти перейшли в активний стан, але тільки видимий було відслідковано. Це коректна поведінка, оскільки ViewPager завжди ініціалізує сусідній до видимого фрагмент.

Також в ході тестування було перевірено коректність спрацювання відправки подій через задані інтервали часу та при досягненні певної кількості подій.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота з розробки та дослідження «Програмний засіб для обробки та передачі аналітичних даних на платформі Android» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

— проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

— розраховано витрати на здійснення науково-технічної розробки;

— розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Програмний засіб для обробки та передачі аналітичних даних на

платформі Android» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [19].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатацій ні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів

Продовження таблиці 5.1

Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Закінчення таблиці 5.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	5	5	5
2. Ринкові переваги (наявність аналогів)	1	1	1
3. Ринкові переваги (ціна продукту)	3	4	3
4. Ринкові переваги (технічні властивості)	3	3	3

Закінчення таблиці 5.2

5. Ринкові переваги (експлуатаційні витрати)	4	4	4
6. Ринкові перспективи (розмір ринку)	2	2	2
7. Ринкові перспективи (конкуренція)	4	4	4
8. Практична здійсненність (наявність фахівців)	4	5	4
9. Практична здійсненність (наявність фінансів)	3	4	4
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	4	3
Сума балів	41	44	41
Середньоарифметична сума балів $СБ_c$	42,0		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [19].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Програмний засіб для обробки та передачі аналітичних даних на платформі Android» становить 42,0 бала, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

5.2 Оцінювання рівня новизни розробки

Виводячи на ринок новинку виробник вважає, що тієї новизни, якою наділена нова розробка є достатньо для того, щоб вона була сприйнята споживачем як нова. Але це не завжди так, в силу того, що споживач і виробник неоднозначно визначать її рівень новизни. Тому доцільним є визначення рівня новизни розробки отриманої в результаті досліджень за темою «Програмний засіб для обробки та передачі аналітичних даних на платформі Android».

Саме визначення рівня і ступеня інтегральної новизни є найбільш актуальним, оскільки її рівень визначає ступінь однакового позитивного сприйняття новизни розробки як виробником, так і споживачем, а отже і ринком в цілому, а це, у свою чергу, є гарантією того, що новинка знайде своє місце на ринку, користуватиметься попитом у споживачів і забезпечить відшкодування витрат, зазнаних товаровиробником під час розроблення та виробництва технічної розробки [20].

Рівень новизни нової продукції розраховуємо експертним методом шляхом протиставлення нової продукції та її аналогів, що існують в даний час на ринку, за чинниками що визначають її значення, в системі «краще-гірше». Рівень новизни встановлюємо відносно рівня аналога (або продукту, що досить близький до аналога).

Для визначення *i*-го виду новизни, застосуємо чинники, які впливають на її рівень. Кожен чинник *i*-го виду новизни розраховуємо в балах. Більша кількість набраних балів свідчить про більший рівень новизни. Для оцінювання рівня новизни використаємо думки експертів, які встановлюють визначені бали відповідним чинникам. Бал відповідності проставляється в діапазоні від (-5 –значно гірше аналога до +5 – значно краще аналога). Результати попереднього оцінювання зведемо до відповідного листа оцінювання (таблиця 5.4).

Таблиця 5.4 – Лист оцінювання рівня новизни експертами

Види та чинники		Бали та експерти		
		Експерт 1	Експерт 2	Експерт 3
1		*2*	*3*	*4*
Споживча новизна	Питома вага 0,225	Максимальний бал $Б_{i\ MAX}$		25
1. Зміна поведінкових звичок споживача		4	4	4
2. Ступінь задоволення потреб і запитів		4	4	4
3. Спосіб задоволення потреби		3	3	3
4. Формування нової потреби		0	0	0
5. Формування нового споживача		0	0	0
Середній бал експертів $Б_{i\ отр}$		11		
Товарна новизна	Питома вага 0,217	Максимальний бал $Б_{i\ MAX}$		30
1. Параметричні зміни показників продукції				
1.1. Якісні		3	3	3
1.2. Технічні		4	4	3
1.3. Економічні		3	3	3
1.4. Сервісні		4	4	4
2. Якість продукції по відношенню до конкурентів		3	3	3
3. Функціональні зміни		3	3	3
Середній бал експертів $Б_{i\ отр}$		20		
Виробнича новизна	Питома вага 0,042	Максимальний бал $Б_{i\ MAX}$		25
1. Рівень унікальності товару для підприємства		5	5	5
2. Рівень унікальності для галузі		3	3	3
3. Рівень унікальності товару для країни		0	0	0
4. Зміна виробничої системи		4	4	4
5. Відносно існуючого асортименту		2	2	2

Продовження таблиці 5.4

Середній бал експертів $Б_{i\ отр}$		14		
Прогресивна новизна	Питома вага 0,179	Максимальний бал $Б_{i\ MAX}$	25	
1. Зміна технології виготовлення		4	4	4
2. Рівень застосування нових компонентів і матеріалів		1	2	1
3. Зміна технологічного принципу дії виробу		1	2	1
4. Зміна конструктивного виконання		3	2	3
5. Рівень застосування інновацій		2	2	2
Середній бал експертів $Б_{i\ отр}$		11		
Ринкова новизна	Питома вага 0,12	Максимальний бал $Б_{i\ MAX}$		20
1. Новий виріб на новому ринку		0	0	0
2. Новий виріб на відомому ринку		2	2	2
3. Модернізований виріб		2	2	2
4. Нова модель		1	2	2
Середній бал експертів $Б_{i\ отр}$		6		
Екологічна новизна	Питома вага 0,035	Максимальний бал $Б_{i\ MAX}$		20
1. Рівень екологічної чистоти технології виробництва		5	5	5
2. Рівень впровадження мало- та безвідходних технологій		5	5	5
3. Рівень екологічно небезпечних режимів експлуатації продукції		5	5	5
4. Рівень забруднення навколишнього середовища		5	5	5
Середній бал експертів $Б_{i\ отр}$		20		
Соціальна новизна	Питома вага 0,036	Максимальний бал $Б_{i\ MAX}$		20
1. Використання нового товару приводить до покращення стану здоров’я нації		0	0	0

Закінчення таблиці 5.4

2. Використання нового товару приводить до зростання доходів населення		0	0	0
3. Виробництво нового товару приводить до збільшення (зменшення) кількості робочих місць на підприємстві		4	5	4
4. Виробництво нового товару приводить до підвищення кваліфікації персоналу		3	3	3
Середній бал експертів $Б_{i\ отр}$		7		
Маркетингова новизна	Питома вага 0,146	Максимальний бал $Б_{i\ MAX}$		20
1. Нові методи маркетингових досліджень		0	0	0
2. Вживання нових стратегій сегментації ринку		1	1	1
3. Вибір нової маркетингової стратегії обхвату і розвитку цільового сегмента		2	3	2
4. Побудова нових каналів збуту		2	2	2
Середній бал експертів $Б_{i\ отр}$		5		

Значення *i*-го виду новизни розрахуємо за формулою [20]:

$$I_i = \frac{Б_{i\,отр}}{Б_{i\,MAX}}, \tag{5.1}$$

де $Б_{i\ отр}$ – отримана кількість балів за шкалою оцінок чинників, що визначають *i*-й вид новизни;

$Б_{i\ MAX}$ – максимальна кількість балів, що може бути отримана за *i*-м видом новизни.

Загальний рівень інтегральної новизни розраховуємо шляхом перемноження отриманого значення *i*-го виду новизни на її вагомість, причому вагомість *i*-го виду новизни визначаємо експертним методом, за формулою [20]:

$$N_{інт} = \sum_{i}^{n} W_i \cdot I_i \,, \qquad (5.2)$$

де $N_{інт}$ – рівень інтегральної (сукупної) новизни;

W_i – вагомість (питома вага) i-го виду новизни;

n – загальна кількість видів новизни.

$$N_{інт} = (0{,}225 \cdot 11/25)+(0{,}217 \cdot 20/30)+(0{,}042 \cdot 14/25)+(0{,}179 \cdot 11/25)+(0{,}12 \cdot 6/20)+$$
$$(0{,}035 \cdot 20/20)+(0{,}036 \cdot 7/20)+(0{,}146 \cdot 5/20) = 0{,}467.$$

Отримане значення інтегрального рівня новизни зіставляємо зі шкалою, що наведена в табл. 5.5 [19].

Таблиця 5.5 – Рівні новизни нового товару та їхня характеристика

Рівні новизни товару	Значення інтегральної новизни	Характеристика товару	Вид нового товару
Найвища	1,00	Абсолютно новий товар	Новий товар, що наділений ознаками інноваційності (інноваційний товар)
Висока	0,8…0,99	Товар, який не має аналогів	
Значуща	0,6…0,79	Принципова зміна споживчих властивостей товару	
Достатня	0,4…0,59	Принципова технологічна модифікація товару	
Незначна	0,2…0,39	Кардинальна зміна параметрів	Новий товар
Помилкова	0,00…0,19	Малоістотна модифікація	

Згідно таблиці 5.5 розробка відповідає рівню при значенні інтегральної новизни 0,467 — достатня новизна; за характеристикою: принципова технологічна модифікація товару; вид розробки — новий товар, що наділений ознаками інноваційності (інноваційний товар).

5.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Програмний засіб для обробки та передачі аналітичних даних на платформі Android», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.3.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [19]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (5.3)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=22 дні.

$$З_о = 13420{,}00 \cdot 22 / 22 = 13420{,}00 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	13420,00	610,00	22	13420,00
Інженер-розробник програмного забезпечення	12920,00	587,27	20	11745,45
Технік	7150,00	325,00	14	4550,00
Всього				29715,45

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Програмний засіб для обробки та передачі аналітичних даних на платформі Android» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i \,, \tag{5.4}$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \,, \tag{5.5}$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо *Мм*=2379,00 грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду [19];

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ – середнє число робочих днів в місяці, приблизно $Т_р$ = 22 дн;

$t_{зм}$ – тривалість зміни, год.

$С_1$ = 2379,00 · 1,10 · 1,65 / (22 · 8) = 24,53 грн

$З_{р1}$ = 24,53 · 5,90 = 144,75 грн

Таблиця 5.7 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка обладнання розробки програмного забезпечення	5,90	2	1,10	24,53	144,75
Підготовка робочого місця розробника програмного забезпечення	4,72	2	1,10	24,53	115,80
Інсталяція програмного забезпечення	5,32	5	1,70	37,92	201,71

Закінчення таблиці 5.7

Компіляція програмних блоків	6,12	4	1,50	33,45	204,74
Налагодження програмних блоків	9,20	6	2,00	44,61	410,38
Тестування Android-систем	14,00	2	1,10	24,53	343,47
Всього					1420,84

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.6)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 12%.

$З_{дод}$ = (29715,45 + 1420,84) · 12 / 100% = 3736,36 грн.

5.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.7)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (29715,45 + 1420,84 + 3736,36) · 22 / 100% = 7671,98 грн.

5.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Програмний засіб для обробки та передачі аналітичних даних на платформі Android».

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot K_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}\ , \qquad (5.8)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

K_j – коефіцієнт транспортних витрат, (K_j = *1,1 ... 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

M_1 = 1,00 · 112,00 · 1,1 - 0,000 · 0,00 = 123,20 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.8 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір Ultra Plus	112,00	1,00	-	-	123,20

Закінчення таблиці 5.8

Папір для записів Light A5	72,00	1,00	-	-	79,20
Органайзер офісний	210,00	1,00	-	-	231,00
Канцелярське приладдя (набір офісного працівника)	174,00	1,00	-	-	191,40
Картридж для принтера Erixon EZ2500	708,00	1,00	-	-	778,80
Диск оптичний NewOPtice CD-RW	12,20	2,00	-	-	26,84
Flesh-пам'ять Kingston 32 GB	243,00	1,00	-	-	267,30
Тека для паперів	87,00	2,00	-	-	191,40
Всього					1889,14

5.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Програмний засіб для обробки та передачі аналітичних даних на платформі Android» відсутні.

5.3.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування

необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ , \qquad (5.9)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, (K_i = *1,10…1,12);*

k – кількість найменувань устаткування.

$B_{спец}$ = 7500,00 · 1 · 1,1 = 8250,00 грн.

Отримані результати зведемо до таблиці:

Таблиця 5.9 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Смартфон Xiaomi POCO X3 NFC	1	7500,00	8250,00
Всього			8250,00

5.3.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.10)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = *1,10…1,12);*

k – кількість найменувань програмних засобів.

$B_{прг}$ = 4280,00 · 1 · 1,1 = 4708,00 грн.

Отримані результати зведемо до таблиці:

Таблиця 5.10 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Середовище розробки Android Studio	1	4280,00	4708,00
Всього			4708,00

5.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \qquad (5.11)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл}$ = (22700,00 · 1) / (2 · 12) = 945,83 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.11 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер розробника програмного забезпечення Macbook Air M1	22700,00	2	1	945,83
Робоче місце інженера-програміста	6750,00	5	1	112,50
Пристрої передачі даних	6720,00	4	1	140,00
Оргтехніка	8100,00	4	1	168,75
Приміщення лабораторії розробки інформаційних систем	200000,00	20	1	833,33
ОС Windows 10	5420,00	2	1	225,83

Закінчення таблиці 5.11

Прикладний пакет Microsoft Office 2016	3750,00	2	1	156,25
Всього				2582,50

5.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (5.12)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 4,50$ грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,45 \cdot 162,0 \cdot 4,50 \cdot 0,95 / 0,97 = 328,05$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.12 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер розробника програмного забезпечення Macbook Air M1	0,45	162,0	328,05

Закінчення таблиці 5.12

Робоче місце інженера-програміста	0,10	162,0	72,90
Пристрої передачі даних	0,03	40,0	5,40
Оргтехніка	0,65	6,2	18,14
Всього			424,49

5.3.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Програмний засіб для обробки та передачі аналітичних даних на платформі Android» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень. Витрати за даною статтєю відсутні.

5.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» відсутні.

5.3.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.13)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$I_в$ = (29715,45 + 1420,84) · 50 / 100% = 15568,15 грн.

5.3.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.14)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (29715,45 + 1420,84) · 100 / 100% = 31136,30 грн.

Витрати на проведення науково-дослідної роботи на тему «Програмний засіб для обробки та передачі аналітичних даних на платформі Android» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.15)$$

$B_{заг}$= 29715,45 +1420,84 +3736,36 +7671,983686 +1889,14 +0,00 +8250,00 + 4708,00 + 2582,50 +424,49 +0,00 +0,00 +15568,15 +31136,30 = 107103,21 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \qquad (5.16)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,95.

ЗВ = 107103,21 / 0,95 = 112740,22 грн.

5.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Програмний засіб для обробки та передачі аналітичних даних на платформі Android» передбачають комерціалізацію протягом 4-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	1000	2500	3500	1000

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 15000 осіб;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 2100,00 грн;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 500,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [19]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.17)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ =35%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2021 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 =$

(500,00·15000,00+2600,00·1000)·0,83·0,35·(10,18/100%)=2405921,00 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 =$

(500,00·15000,00+2600,00·3500)·0,83·0,35·(10,18/100%)=3954286,00 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 =$

(500,00·15000,00+2600,00·7000)·0,83·0,35·(10,18/100%)=6121997,00 грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4 =$

(500,00·15000,00+2600,00·8000)·0,83·0,35·(10,18/100%)=6741343,00 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.18)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ=0,14;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП$ =2405921,00/(1+0,14)^1+3954286,00/(1+0,14)^2+6121997,00/(1+0,14)^3+

+6741343,00/(1+0,14)^4=2110457,02+3042694,68+4132173,60+3991416,23=13276741,52 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (5.19)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=1,5;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 112740,22 грн.

$PV = k_{інв} \cdot ЗВ$ = 1,5 · 112740,22 = 169110,33 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{5.20}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 13276741,52 грн;

PV – теперішня вартість початкових інвестицій, 169110,33 грн.

$E_{абс} = ПП - PV$ = 13276741,52 - 169110,33 = 13107631,20 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.21}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 13107631,20 грн;

PV – теперішня вартість початкових інвестицій, 169110,33 грн;

$T_{ж}$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$$E_{в} = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+13107631,20/169110,33)^{1/4} = 1,98.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \tag{5.22}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2021 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,16.

$\tau_{мін}$ = 0,11+0,16 = 0,27 < 1,98 свідчить про те, що внутрішня економічна дохідність інвестицій $E_{в}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Програмний засіб для обробки та передачі аналітичних даних на платформі Android» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_{в}}, \tag{5.23}$$

де $E_{в}$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 1,98 = 0,51 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Висновки до розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Програмний засіб для обробки та передачі аналітичних даних на платформі Android» становить 42,0 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

Також термін окупності становить 0,51 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Програмний засіб для обробки та передачі аналітичних даних на платформі Android».

ВИСНОВКИ

В даній магістерській кваліфікаційній роботі створено програмний засіб для обробки та передачі аналітичних даних на системі Android. Даний додаток призначений для застосування в коді клієнтського додатку та дозволяє проводити збір аналітичних даних з мінімальною кількістю необхідного коду.

В першому розділі роботи було проведено аналіз сучасного стану в сфері обробки аналітичних даних у мобільних додатках та проведений огляд сучасних бібліотек для збору аналітичних даних.

В другому розділі розроблено удосконалений метод збору та передачі аналітичних даних у клієнтських додатках, що працюють під операційною системою Android. Описано підходи та технології, що необхідно застосувати для реалізації даного методу, такі як аспектно-орієнтоване програмування, підхід з використанням корутин, який спрощує роботу з фоновими потоками в додатках з мережевими викликами та роботу з файлами, реалізація відслідковування зміни сторінок з можливістю виклику в активний момент життєвого циклу.

Для реалізації автоматичного генерування подій були використані принципи аспектно-орієнтованого підходу з використанням бібліотеки AspectJ. Відслідковування зміни екранів виконано за допомогою інтеграції з такими компонентами системи Android як Jetpack Navigation та ViewPager. Додаток розроблено з використанням останніх підходів до створення Android додатків, а саме мови програмування Kotlin та підтримкою Kotlin Coroutines.

Створений додаток відповідає стандартам регламенту GDPR та не порушує права користувача на обробку персональних даних. Проведені економічні розрахунки вказують на доцільність розробки.

В третьому та четвертому розділах роботи наведено програмну реалізацію додатку та результати його тестування, відповідно. Розроблено Gradle плагін для підтримки АОП. В процесі тестування було перевірено коректність роботи додатку, а саме автоматичне відслідковування зміни

екранів та автоматичне генерування подій з подальшою фільтрацією параметрів на базі вмісту конфігурації.

В п’ятому розділі роботи проведено економічне обгрунтування доцільності розробки та виконані розрахунки економічної ефективності запропонованого рішення. Результати свідчать про високий рівень комерційного потенціалу розробки. Термін окупності становить 0,51 р., що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Молодь в науці: дослідження, проблеми, перспективи. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/view/14231
2. Personalisation can lift push notification open rates by up to 800%. URL: https://econsultancy.com/personalisation-can-lift-push-notification-open-rates-by-up-to-800-study/ (дата звернення: 05.09.2021).
3. Филлипс Б., Стюарт К., Марсикано К. Android. Программирование для профессионалов. 3-е издание. Спб, 2019. 67 с.
4. What is GDPR, the EU's new data protection law? URL: https://gdpr.eu/what-is-gdpr/ (дата звернення: 18.09.2021).
5. Firebase Documentation. URL: https://firebase.google.com/docs (дата звернення: 22.10.2021).
6. Segment for Developers. URL: https://segment.com/docs/guides/intro-impl/ (дата звернення: 24.10.2021).
7. Эккель Б. Философия Java 4-е изд. Спб, 2019. 107 с.
8. Aspect Oriented Programming with Spring. URL: https://docs.spring.io/spring-framework/docs/2.5.x/reference/aop.html (дата звернення: 09.10.2021).
9. AspectJ Documentation packages. URL: https://www.eclipse.org/aspectj/docs.php (дата звернення 26.10.2021).
10. Фрімен Е. Патерни проектування. Харків, 2020. 318 с.
11. Исакова С., Жемеров Д. Kotlin в действии. Москва, 2017. 82 с.
12. Coroutines guide. URL: https://kotlinlang.org/docs/coroutines-guide.html (дата звернення: 30.09.2021).
13. Get started with the Navigation component URL: https://developer.android.com/guide/navigation/navigation-getting-started (дата звернення: 15.10.2021).

14. Slide between fragments using ViewPager. URL: https://developer.android.com/training/animation/screen-slide (дата звернення: 18.10.2021).

15. LiveData Overview. URL: https://developer.android.com/topic/libraries/architecture/livedata (дата звернення: 05.11.2021).

16. Gradle Kotlin DSL Primer. URL: https://docs.gradle.org/current/userguide/kotlin_dsl.html (дата звернення: 06.11.2021).

17. Developing Custom Gradle Plugins. URL: https://docs.gradle.org/current/userguide/custom_plugins.html (дата звернення: 11.11.2021).

18. What is Context in Android and which one should you use? URL: https://medium.com/@banmarkovic/what-is-context-in-android-and-which-one-should-you-use-e1a8c6529652 (дата звернення: 02.11.2021).

19. Козловський В. О., Лесько О. Й., Кавецький В. В. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. Вінниця : ВНТУ, 2021. 42 с.

20. Кавецький В. В. Економічне обґрунтування інноваційних рішень. Вінниця : ВНТУ, 2016. 113 с.

ДОДАТОК А

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ ВНТУ

д.т.н., проф.

_____О. Д. Азаров

“__” _____ 2021 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

«Програмний засіб для обробки та передачі аналітичних даних на платформі Android»

08-23.МКР.013.00.000 ПЗ

Науковий керівник к.т.н., доц. каф. ОТ

_____ Войцеховська О. В.

Студент групи 1КІ-20М

_____ Чурчун В. В.

Вінниця 2021

Підставою для виконання МКР є потреба в покращенні методу обробки та відправки аналітичних даних.

Мета проекту — розробка програмного засобу для обробки та передачі аналітичних даних на платформі Android;

Призначення розробки — полегшити збір та обробку аналітичних даних у додатках на операційній системі Android.

Джерелами розробки МКР є Інтегроване середовище розробки Android Studio та мова програмування Kotlin.

Вимогами до виконання МКР є наявність програмного засобу та його відповідність функціональності.

Етапи МКР та очікувані результати наведені в таблиці А.1.

Таблиця А.1 – Етапи виконання роботи

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Формулювання мети та цілей для її досягнення	07.09.21	10.09.21	Вступ
2	Аналіз сучасного стану в сфері аналітики даних та огляд існуючих бібліотек для збору аналітичних даних	11.09.21	30.09.21	Розділ 1
3	Огляд методів та технологій для збору й обробки аналітичних даних	01.10.21	30.10.21	Розділ 2
4	Написання додатку	01.11.21	15.11.21	Розділ 3
5	Тестування створеного додатку	16.11.21	19.11.21	Розділ 4
6	Економічний аналіз роботи	20.11.21	28.11.21	Розділ 5
7	Оформлення додатків	29.11.21	30.11.21	Додатки
8	Оформлення пояснювальної записки	01.12.21	05.12.21	ПЗ

До захисту МКР надаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі,

відзив наукового керівника, відзив опонента, анотації до МКР українською та іноземною мовами.

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

Вимоги до оформлення МКР викладені в МЕТОДИЧНИХ ВКАЗІВКАХ до дипломного проектування, ДСТУ 3008-2015 та ДСТУ 3974-2000 «Правила виконання дослідно-конструкторських робіт. Загальні положення».

Технічне завдання до виконання отримав _____ Чурчун В. В.

ДОДАТОК Б

Частота відкриття додатків з персоналізованими та не персоналізованими даними

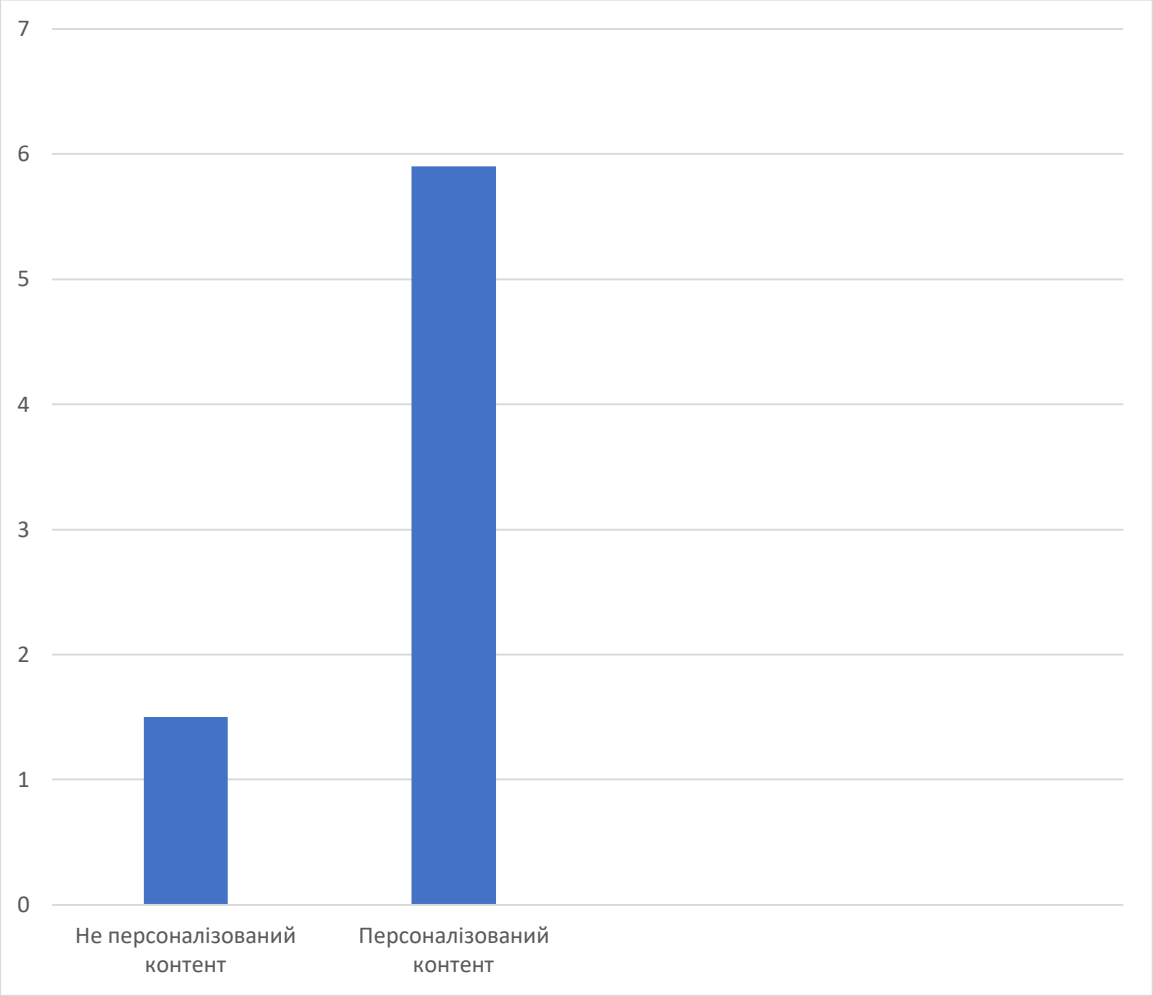

Рисунок Б.1 – Частота відкриття додатків з персоналізованими та не персоналізованими даними

ДОДАТОК В

Відсоткова залежність користувачів, що відмовляються від додатків після першого використання

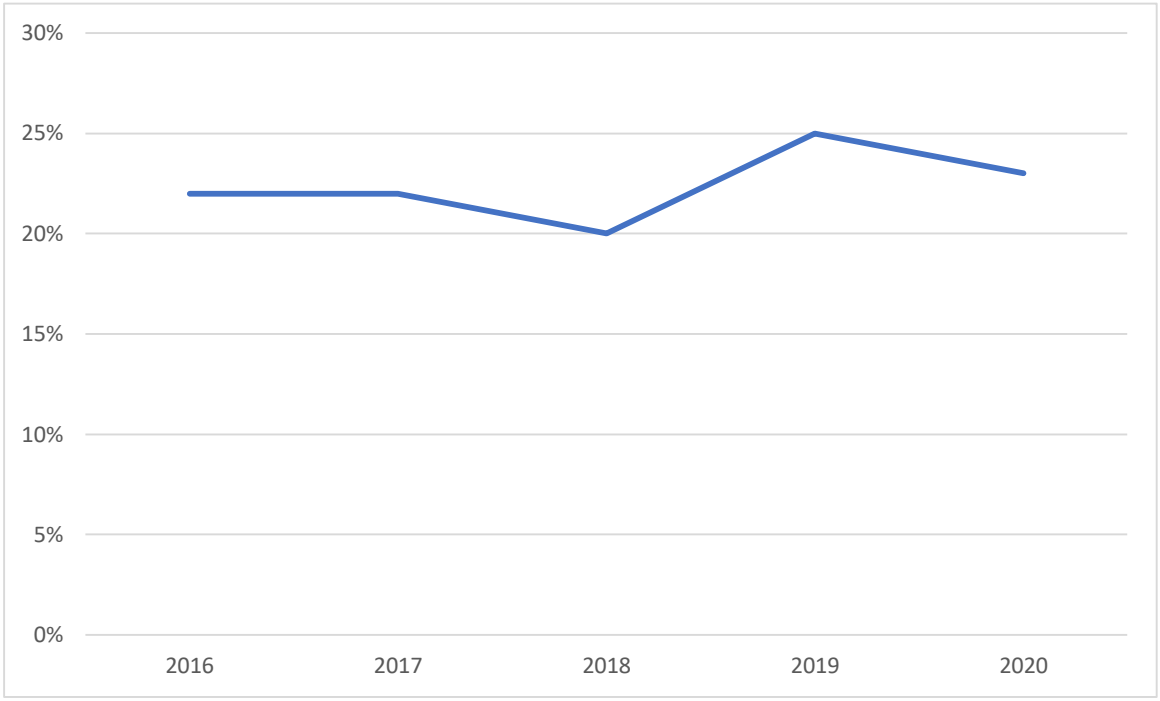

Рисунок В.1 – Відсоткова залежність користувачів, що відмовляються від додатків після першого використання

ДОДАТОК Г

Візуалізація роботи вдосконаленого методу збору й обробки аналітичних даних

Рисунок Г.1 – Візуалізація роботи вдосконаленого методу збору й обробки аналітичних даних

ДОДАТОК Д

Життєвий цикл фрагмента у системі Android

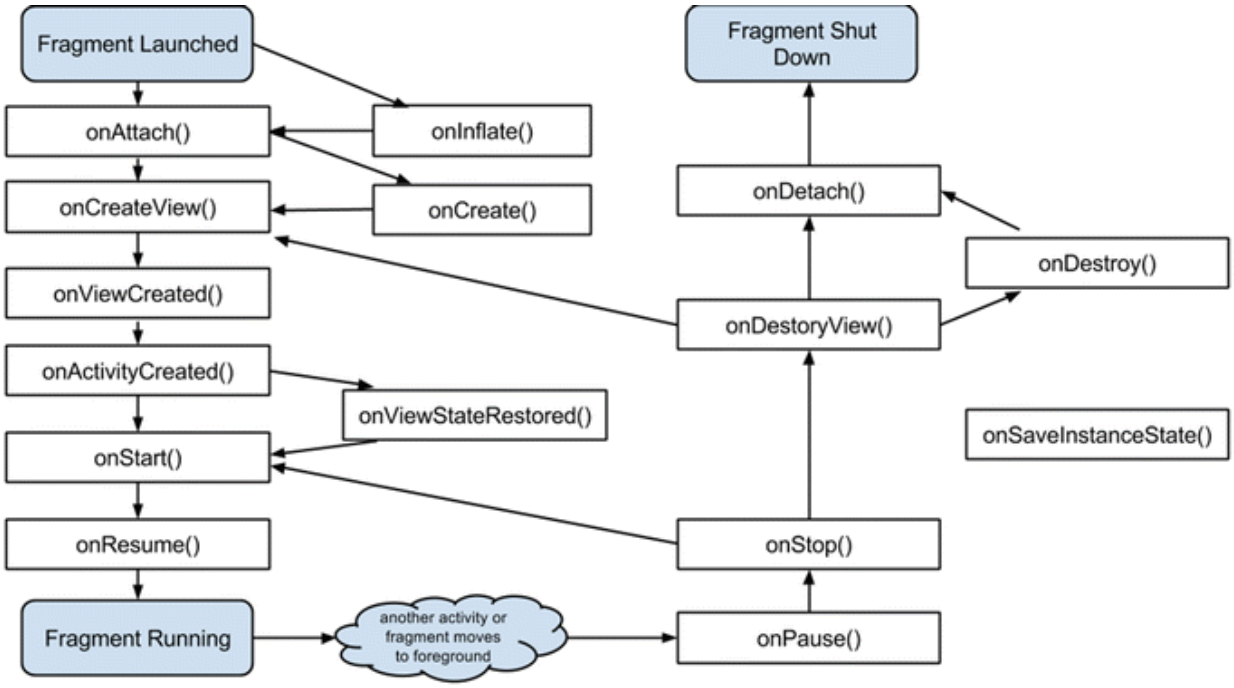

Рисунок Д.1 – Життєвий цикл фрагмента у системі Android

ДОДАТОК Е

UML діаграма додатку

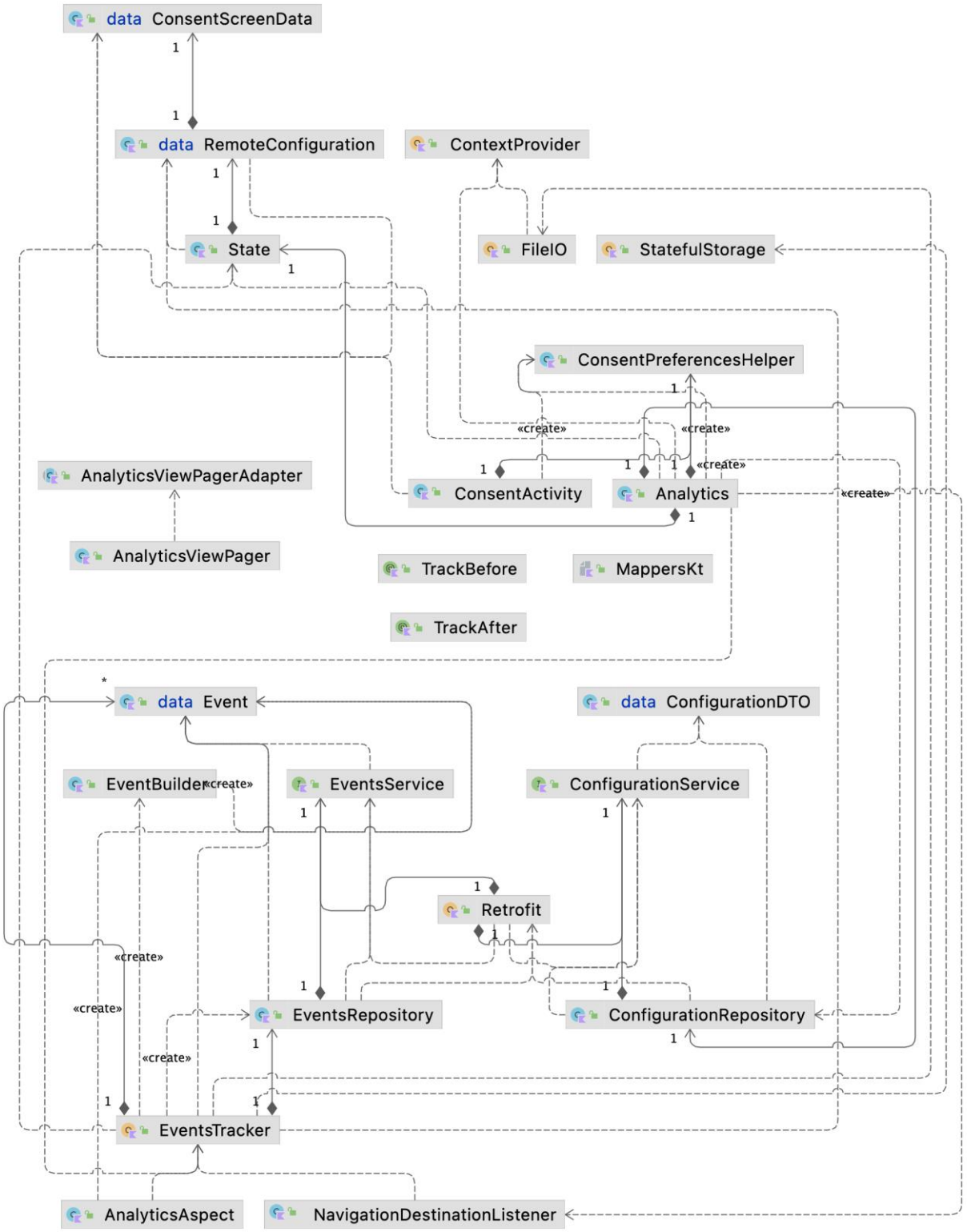

Рисунок Е.1 – UML діаграма додатку

ДОДАТОК Ж

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Програмний засіб для обробки та передачі аналітичних даних на платформі Android

Тип роботи: магістерська кваліфікаційна робота

(кваліфікаційна роботи, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))

Підрозділ кафедра обчислювальної техніки, ФІТКІ, гр. 1КІ-20м

(кафедра, факультет (інститут), навчальна група)

Науковий керівник Войцеховська О. В., доцент кафедри обчислювальної техніки

(прізвище, ініціали, посада)

Показники звіту подібності

Plagiat.pl (StrikePlagiarism)		Unicheck	
КП1			
КП2		Оригінальність	95,8
Тривога/Білі знаки	/	Схожість	4,2

Аналіз звіту подібності (відмінити подібне)

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності і відсутності самостійності її автора. Робот направити на доопрацювання.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений(-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Чурчун В. В.

(підпис) (прізвище, ініціали)

Опис прийнятого рішення

_Ступінь оригінальності роботи відповідає вимогам, що висуваються до МКР

_____

_____

Особа, відповідальна за перевірку_____ _____

(підпис) (прізвище, ініціали)

Експерт _____ _____

(за потреби) (підпис) (прізвище, ініціали)